Embedded Real-Time H264/AVC High Definition Video Encoder on TI's KeyStone Multicore DSP

Nejmeddine Bahri, Thierry Grandpierre, Med Ali Ben Ayed, Nouri Masmoudi, Mohamed Akil

Embedded real-time H264/AVC high definition video encoder on TI's KeyStone multicore DSP

Nejmeddine Bahri[1], Thierry Grandpierre[1], Med Ali Ben Ayed[2], Nouri Masmoudi[2], Mohamed Akil[1]
[(1)]ESIEE Engineering, LIGM Laboratory, University Paris-EST, France
[(2)]National school of Engineers, LETI Laboratory, University of Sfax, Tunisia
nejmeddine.bahri@esiee.fr thierry.grandpierre@esiee.fr nouri.masmoudi@enis.rnu.tn

Abstract –To overcome high computational complexity of advanced video encoders for emerging applications that require real-time processing, using multicore technology can be one of the promising solutions to meet this constraint. In this context, this paper presents a parallel implementation of the H264/AVC high definition (HD) video encoder exploiting the power processing of eight-core digital signal processor (DSP) TMS320C6678. GOP Level Parallelism approach is used to improve the encoding speed and meet the real-time encoding compliant. A master core is reserved to handle data transfer between the DSP and the camera interface via a Gigabit Ethernet link. Multithreading algorithm and ping-pong buffers technique are used to enhance the classic GOP level parallelism approach and hide communication overhead. Experimental results on seven slave DSP cores, running each at 1 GHz, show that our implementation allows performing a real-time HD (1280x720) video encoding. The achieved encoding speed is up to 28 f/s. The proposed parallel implementation accelerates the encoding process by a factor of 6.7 without inducing quality degradation in terms of PSNR or bit-rate increase compared to single core implementation. Experiments show that our proposed scheduling technique for hiding communication overhead allows saving up to 36% of the fully encoding chain time which includes frames capturing, frames encoding and bitstream saving in a file.

Keywords*: H264/AVC encoder, real-time, multicore DSP, GOP Level Parallelism.*

I. Introduction

Nowadays, embedded processors occupy the majority of multimedia systems such as smart cameras, digital TV, Smartphone, and video surveillance platforms. In the other side, facing the rapid evolution of digital cameras technology, HD resolution becomes widely used in several multimedia applications in order to ensure better video quality. Consequently, video encoding with high compression performance is required to overcome the huge amount of data transmission, memory storing requirement, and transmission bandwidth limitation.

H264/AVC [1] encoder represents one of the most efficient video standards. It is characterized by a better video coding efficiency compared to previous ones. However, this efficiency is followed by a high computational complexity that requires a high-performance processing capability to meet the real-time constraint of 25 f/s. Moreover, this complexity is drastically increased with HD resolution which makes it hard to achieve real-time encoding with low frequency processors.

With the fast evolving of embedded processors technology in terms of high processing frequency and multicore architectures, developers become actually able to perform more complex applications that require real-time processing and high computing performance. In fact, multicore technology allows overcoming the frequency limitation of mono-core processors and makes it possible to process several tasks simultaneously with the minimum of power consumption. In this context, many researchers have been conducted on the parallelism of H264/AVC video encoder on multicore platforms. Different partitioning techniques have been discussed in order to accelerate the encoding process and meet the real-time encoding compliant.

In this context, this paper presents an optimized H264/AVC HD video encoder implementation on a multicore DSP TMS320C6678. GOP Level Parallelism approach is used to accelerate the encoding speed. A real-time video coding demo is described taken into account image capture from a camera interface, DSP encoding, and bitstream saving in a file. Our implementation is enhanced by performing a multi-threading algorithm and exploiting the standard ping-pong buffers technique in order to hide communication overhead.

The remainder of this paper is outlined as follows: next section presents the different partitioning methods for H264/AVC video encoder and discusses some parallel implementations of this standard. TMS320C6678 multicore DSP architecture is described in section 3. Section 4 details our proposed implementation based on the classic GOP Level Parallelism approach which is performed on seven slave DSP cores. It highlights also our video encoding demo including image capture and bitstream saving. This section is then concluded by discussing the achieved encoding performances. The

enhanced GOP Level Parallelism approach, based on optimizing communication overhead, is detailed in section 5. At the end of this section, experimental results are presented and discussed. Finally, section 6 concludes this paper and presents some prospects.

II. Partitioning approaches and related works for the H264/AVC video encoder

II.1. Partitioning approaches

To profit from the potential parallelism in H264/AVC encoder, two mainly techniques could be exploited to parallelize this encoder on a multicore platform:

II.1.1. Task-level parallelization (TLP)

This approach splits the encoder into several tasks, equal to the number of threads available on the system and run these tasks simultaneously as a pipeline. Consequently, we have to choose the appropriate functions that should be grouped together to be processed in parallel and the other functions that will be executed in serial to respect data dependencies. Furthermore, tasks computational complexity should be considered to maximize the encoding speedup and ensure a workload balance for the different tasks. Finally, when grouping functions, synchronization overhead should be minimized as much as possible by eliminating data dependency between the different function blocks.

II.1.2. Data-level parallelization (DLP)

This technique exploits the hierarchical data structure of H264/AVC encoder by simultaneously processing several data levels on multiple processing units. In fact, H.264/AVC encoder baseline profile splits a video sequence into an hierarchical structure as shown in Fig .1. The video sequence consists of one or more groups of pictures (GOP). Each GOP includes one or more frames and always starts with intra frame (I) where only the intra prediction [1] is performed to reduce the spatial redundancy. The remaining frames are a predicted frames (P) where both intra and inter predictions [1] are performed to reduce both spatial and temporal redundancies respectively. Finally, the frames are also divided into one or more slices, subdivided themselves into macroblocks (MB) and blocks.

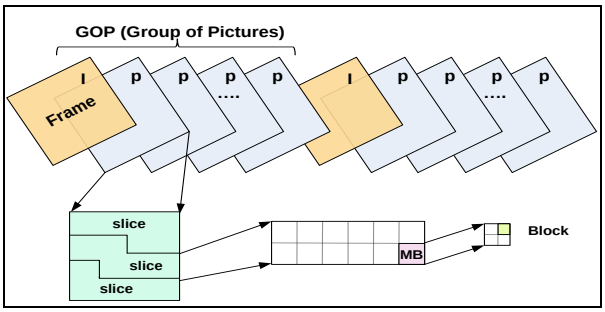

Fig. 1. Hierarchical decomposition of a video sequence in H264/AVC standard

DLP is restricted by data dependencies among different data units (spatial dependencies in the same frame required for the intra prediction and temporal dependencies between successive frames required for the inter prediction).
According to data structure of H264/AVC encoder, several parallelism approaches can be applied such as:

GOP Level Parallelism: several GOPs can be encoded in parallel as no dependencies exist among different GOPs. In fact, each GOP starts with intra frame "I frame" where the current MB intra encoding requires only data from its neighboring MBs. This approach is characterized by high performance scalability, thus speedup is enhanced as the number of available cores is increased. This technique does not require a high synchronization cost and does not induce any rate distortion. However, this approach requires a high memory amount to handle all the GOP frames which makes it not well suitable for System on Chip (SOC) platforms as the Chip surface is an important factor for the design evaluation.

Frame Level Parallelism: a partial dependency between successive "P frames" in the same GOP is existed. In fact, to calculate the current MB motion vector, a motion estimation algorithm is performed in a specific area named "search window" in the reference frames (previously encoded) as shown in Fig. 2. Consequently, several frames can be encoded in a pipeline way as well as the search window is already encoded.

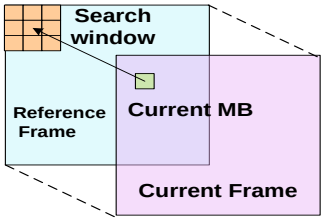

Fig. 2. Data dependency for inter prediction

Slice Level Parallelism: H264/AVC standard gives the choice to split the frame into independent slices. Thus, several slices can be processed in parallel in function of the number of available threads or processing units. This approach presents high performance scalability. No synchronization cost among threads is needed. This approach does not require a lot of memory amount. However, slice level parallelism technique induces bit-rate degradation because intra dependencies are not respected for TOP MBs of each slice.

MB Level Parallelism: in the frame itself, several MBs can be processed in parallel once their neighboring MBs (TOP, TOP Left, Top Right and Left) are already encoded to respect intra data dependency. This approach is characterized by low encoding latency and does not require a lot of memory amount. However, low performance scalability and high synchronization cost are the major drawbacks of this approach.

II.2. Related works

To overcome the high complexity of H264/AVC video encoder, several works have exploited the parallelism in H264/AVC encoder to meet the real-time encoding compliant and achieve a good encoding speedup which can be presented by the following equation (1).

$$\textbf{speedup} = \frac{\textbf{time of sequential encoding}}{\textbf{time of parallel encoding}} \quad (1)$$

Several implementations exploiting multi-thread, multiprocessor, and multicore architectures are discussed in many papers. Different methods of partitioning have been applied as:

II.2.1. Task Level parallelism approach

Several works have applied TLP approach to accelerate the encoding run-time. As examples, we note:

Zhibin Xiao et al [2] mapped the dataflow of H.264/AVC encoder on 167-core asynchronous array of simple processors (AsAP) computation platform. They processed the luminance and the chrominance components in parallel. Intra4x4 modes and intra16x16 modes are processed in parallel. Only 3 modes for intra4x4 instead of 9 and 3 modes for intra16x16 are considered to reduce the top right dependency. Eight processors are used for ICT (Integer Cosine Transform) and quantification. Seventeen processors are reserved for CAVLC (Context Adaptive Variable Length Coding) entropy coding and a hardware accelerator is used for motion estimation. Despite all these optimizations, the real-time is not achieved. The presented encoder is able to encode only 21 f/s for VGA resolution (640 x 480). Also, reducing the number of candidate modes for intra16x16 and intra4x4 affects the visual quality in terms of PSNR and induces a bit-rate increase.

Ming-Jiang Yang et al [3] implemented the H264/AVC encoder on a dual-core DSP processor ADSP-BF561 chipset using functional partitioning. Core A of BF561 processor is devoted to perform mode decision, motion compensation, intra prediction, integer transform (IT), quantization, de-quantization, inverse integer transform, and entropy encoding. Core B is dedicated to perform boundary extension, in-loop filtering, and half-pel interpolation. Core A and core B perform tasks in two pipeline stages. The proposed system achieves real-time encoding for CIF format (352x288) but not yet for higher resolutions (SD (720x480) and HD (1280x720).

Seongmin Jo et al [4] used OpenMP programming model to parallelize H264/AVC encoder with the TLP approach. They executed motion estimation modes (16x16, 16x8, x8x6 and 8x8), intra prediction modes (intra4x4 and intra16x16) and de-blocking filter for the previous MB in parallel as seven different tasks on ARM Quad MPCore platform. Mode decision and entropy coding are processed thereafter in serial. The obtained speedup on 4 cores is 1.67 which looks not significant compared to the number of used cores. This is justified by that OpenMP programming model is a generic parallelization approach which is not optimized for specific applications such as H264/AVC encoder.

Hajer et al. [5] presented a high level parallelization approach to develop an optimized parallel model of H264/AVC encoder for embedded SoCs. The proposed implementation is based on the exploration of task level parallelism and the use of the parallel Kahn process network (KPN) model of computation, and the YAPI programming C++ runtime library. They partitioned the H264/AVC encoder into several tasks and run each task on a processor. Moreover, they divided the motion estimation and compensation modules into 3 processes to accelerate their processing. Experiments on a SOC platform including 4 MIPS processors show that a speedup of 3.6 is achieved for QCIF format but real-time encoding performance is still not achieved (7.7 f/s for QCIF format).

Although task level partitioning ensures low encoding latency, it has some drawbacks. In fact, it provides less encoding efficiency due to a lot of data dependencies among tasks that require a large amount of data transfers among processors; thus, consumption of the system bandwidth. Moreover, functions in H.264/AVC encoder have not the same load balance which makes it hard to uniformly map functions among processors. As a result, the final performance is always limited by the heaviest load processor. Furthermore, for some multicore platforms, cache memory is not automatically coherent as it is for general purpose multiprocessor. This may impose cache coherence problems [6] when a core reads a shared data which was processed by another core. Consequently, cached data write-back and invalidation cache lines are required for each data reading and writing among the different cores. These instructions relatively reduce the implementation efficiency especially when there are a lot of data dependencies.

II.2.2. GOP Level parallelism approach

GOP level parallelism approach was applied in several works to benefit from the high performance scalability and low synchronization cost.

S.Sankaraiah et al. [7] [8] applied this technique and exploited a multithreading algorithm for H264/AVC encoder. Each GOP is handled by a separate thread. Frames in each GOP are encoded by two threads: I and P frames by the first thread and B frames by the second thread. The obtained speedup using dual and quad core processors are 5.6 and 10 respectively. Real-time is not noticed in the paper.

Rodriguez et al. [9] proposed an H264/AVC encoder implementation using GOP level parallelism combined with slice level parallelism. They used a clustered workstations solution with the MPI technique (Message Passing Interface) to ensure data communications among the different processors. The drawback of this solution is that clustered workstations are a costly solution and they are not devoted for embedded applications. Moreover, a bit-rate increase is noticed owing to the use of slice level parallelism technique.

Fang Ji et al. [10] implemented this encoder on an MPSOC platform (Multi-Processor System On Chip) using GOP level parallelism technique. They build three Microblaze soft cores based on XILINX FPGA. A master processor is devoted to allocate frames into the shared memory. Thus, each slave coprocessor encodes its appropriate GOP. Experiments show that the obtained speedup is 1.831. The real-time encoding compliant is not achieved. In fact, this solution can encode only 3 f/s for QCIF (176x144) resolution.

II.2.3. Frame Level Parallelism approach

Generally, frame level parallelism approach is combined with another technique as it is mentioned in Zhuo Zhao's paper [11]. Authors proposed a new wave-front parallelization method for H264/AVC encoder. They mixed two partitioning methods: frame level parallelism and MB row level parallelism. All the MBs in the same row are processed by the same processor or thread to reduce data exchanges among processors. MBs in different frames can be processed concurrently once the search window is already encoded. A Pentium 4 processor running at 2.8 GHz is used to run the H264/AVC Joint Model software (JM) 9.0 [12]. Simulations on 4 processors show that a speedup factor of 3 is achieved (3.17 QCIF and 3.08 for CIF). However, real-time encoding is not reached and only 1frame/1.72s is encoded for CIF resolution.

II.2.4. slice Level parallelism approach

Slice level parallelism approach was applied in several works to profit from low synchronizations among different threads or processors.

Yen-Kuang et al. [13] used OpenMP programming model to accelerate the encoding process. Slice level partitioning is performed on 4 Intel Xeon™ processors with Hyper-Threading Technology. Simulations show that a speedup factors ranging from 3.74 to 4.53 are obtained on 4 processors with Hyper-threading technology for CIF and SD resolutions.

Olli lehtoranta et al. [14] implemented a row-wise data parallel video coding method on quad TMS320C6201 DSP system. The frame is divided into slices by row-wise and each slice is processed by one DSP. A DSP master is devoted to swap data to/from the remaining DSPs. Real-time encoding performance (30f/s) is achieved only for CIF resolution but not yet for higher resolutions.

Despite its simplicity, slice level parallelism approach has two mainly drawbacks. First, it provides quality degradation in terms of PSNR and second, it induces an important increase in bit-rate especially when splitting the frame into a great number of slices.

II.2.5. MB Level parallelism approach

In several papers, MB level parallelism is used to accelerate the encoding speed and ensure a low encoding latency with the minimum of memory requirement.

Sun et al. [15] implemented a parallel algorithm for H264AVC encoder based on a MB region partition (MBRP). They split the frame into several MB regions composed by adjoining columns. Then, they mapped the MB regions onto different processors to be encoded. Data dependencies in the same MBs row are all respected. Simulation results on 4 processors running at 1.7 GHz show that the proposed partitioning achieves a speedup factor of 3.33 without inducing the rate distortion compared to the software JM10.2. However, real-time is not achieved. In fact, the achieved encoding speed is only 1frame/1.67s for CIF resolution and 1frame/6.73s for SD resolution.

Shenggang Chen et al. [16] introduced an on-chip parallel H264/AVC encoder implementation on hierarchical 64-cores DSP platform. This platform consists of 16 super nodes (4 DSP cores for each node). 2D Wave-front algorithm is used and each MB is assigned to one super node. Subtasks for encoding one MB such as motion estimation, intra prediction, and mode decision are further parallelized to keep busy the four DSP cores that form a node. Speedup factors of 13, 24, 26 and 49 are achieved for QCIF, SIF (352x240), CIF, and HD sequences respectively. The proposed wave-front parallel algorithm does not introduce any quality loss. However, the use of CABAC-based bit-rate estimation and the parallel CABAC evolutional entropy coder cause a bit-rate increase. Real-time processing is not indicated in this paper.

Seongmin Jo et al [4] applied as second method, a DLP approach with OpenMP programming model. 2D wave-front approach is performed by processing several MBs in the same frame in parallel way. Experiments on an ARM MPCore platform show that this approach achieves a speedup factor of 2.36 using 4 threads. However, real-time encoding performance is not noticed also in this work.

II.2.6. Combined approaches

Multiple partitioning methods are combined in several works to ensure more encoding efficiency and well exploit the H264/AVC encoder parallelism.

Huayou Su et al [17] proposed a parallel framework for the H264/AVC encoder based on a massively parallel architecture implemented on NVIDEA's GPU (Graphic Processor Unit) using CUDA (Compute Unified Device Architecture). They presented several optimizations to accelerate the encoding speed. A parallel implementation of the inter prediction is proposed based on a novel algorithm MRMW (Multi-resolutions Multi-windows) that consists in using the motion trend of a lower resolution frame to estimate that of the original frame (higher resolution). MRMW steps are parallelized with CUDA on different GPU's cores. In addition, they performed a multi-level parallelism for intra-coding by performing a multi-slice encoding. Each frame is partitioned into independent slices and the wave-front method is adopted to parallelize MBs encoding in the same slice. Some dependencies within MBs are not respected in order to maximize the parallel processing.

Moreover, CAVLC coding and filtering processes are also parallelized by dividing these modules into several tasks. Experimental results show that a speedup factor of 20 is obtained. The presented parallel H264/AVC encoder is able to perform a real-time HD video encoding. However, this implementation affects the visual quality by inducing a PSNR degradation ranging from 0.14 dB to 0.77 dB and induces a little increase in bitrate.

António Rodrigues et al. [18] implemented the H264/AVC encoder on a 32-core Non-Uniform Memory Access (NUMA) computational architecture with eight AMD 8384 quad-core chip processors running at 2.7 GHz. Two parallel levels are combined: slice level and MB level. A multi-threading algorithm using openMP is used for the JM software. The frame is divided into several slices and each slice is processed by a group of cores. Several MBs in the same slice are encoded in parallel. Data dependencies are respected by the different cores of the group. The achieved speedup using the whole set of 32 cores is between 2.5 and 6.8 for 4CIF video (704 × 576). These speedups are not significant compared to the number of exploited cores. Using a MB level parallelism requires that data must be shared which may lead to a memory bottleneck and higher latency. Also, increasing the number of slices induces a bit-rate distortion. Finally, real-time encoding performance is not noticed in this work.

III. DSP platform description

New DSP processors represent a promising solution for high performance applications and embedded systems implementations. They are characterized by a software flexibility, high performance computing, low power consumption, competitive price tag, and time to market. VLSI (Very Large Scale Integration) technology evolution allows designing new generations of DSP processors characterized by high frequency processing and multicore architecture. This gives more efficiency to DSP platforms and motivates developers to perform more complex applications that require high performance computing. For these reasons, we chose to implement the H264/AVC encoder on the multicore DSP TMS320C6678 to profit from its new features to meet the real-time encoding compliant (25 f/s). TMS320C6678 DSP [19] belongs to the latest generation of multicore DSPs made by Texas Instrument (TI). It is the highest-performance fixed/floating-point DSP which is based on TI's KeyStone multicore architecture. This platform includes multicore navigator, teraNet, hyperlink, and multicore shared memory controller which provide adequate internal bandwidth for non-blocking access to all processing cores, coprocessors, peripherals, and I/O. As presented in Fig. 3, eight TMS320C66x DSP Core Subsystems (C66x CorePacs) running each at 1GHz, very long instruction word (VLIW) architecture, Single Instruction Multiple Data (SIMD) set instruction and 8.5 Mega-bytes (Mb) of memory on chip are combined to deliver 64000 MIPS performance. Each C66x DSP core integrates a large amount of on-chip memory. In addition to 32 Kilo-bytes (KB) of L1 program and data cache, 512 KB of internal memory per core that can be configured as mapped RAM or cache is integrated on the chip. The platform also integrates 4Mb of shared memory. To support applications that require a large amount of memory such as ultra HD video applications, TMS320C6678 includes 512 Mb of DDR3-1333 external memory. This platform comes with the TI's Multicore Software Development Kit (MCSDK) for SYS/BIOS Real-Time Operating System (RTOS). Performance is also enhanced by an EDMA controller (Enhanced Direct Memory Access) which is able to manage memory transfers independently from the CPU. TMS320C6678 supports several high speed standard interfaces including RapidIO for DSP-to-DSP communications, PCI Express Gen2, and Gigabit Ethernet for Internet Protocol (IP) networks. It also includes I^2C, UART and Telecom Serial Interface Port (TSIP).

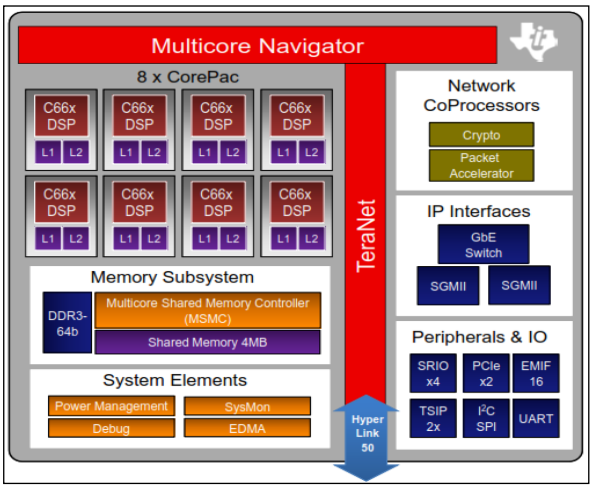

Fig. 3. Internal architecture of TMS320C6678 multicore DSP

IV. The Classic multicore implementation of the GOP Level Parallelism approach

IV.1. Partitioning strategy

From previous works, several partitioning methods have been applied and each of them has as advantages as also some drawbacks. Choosing the appropriate partitioning approach depends also on the target platform. In fact, the number of available processing units and processors inter communication medium (shared memory, point to point, FIFO memory, message passing interface) should be necessarily taken into account to perform an efficient parallel implementation. Consequently, several points should be considered:

- Memory constraint is not an important factor for DSP-based implementations as well as for SOC platforms. Consequently, this gives more liberty to choose a partitioning method.

- Since C6678 DSP memory cache is not automatically coherent, a simple partitioning method which does not require an important inter-processors communications and characterized by a low synchronization cost should be selected. This allows reducing the required write-backs and cache lines invalidations to avoid cache coherence problem.
- The selected method should not affect the bit-rate performance or the visual quality and preferably characterized by a high encoding scalability.

Based on these points, we adopt the GOP level parallelism approach to achieve real-time encoding processing without inducing any rate distortion. In fact, no dependencies among GOPs make this approach easy for implementation. It does not require data transfers or synchronizations among processors and characterized by a high encoding scalability. Finally, cache coherence problem with GOP level parallelism method is almost abolished compared to TLP approach.

IV.2. System platform description

To perform a real-time video encoding demo, frames acquisition by the DSP platform should be also performed in real-time. Consequently, 277Mbits/s as transmission bandwidth (35 Mbytes/s) at least is required to transfer HD frames (1280x720) at 25 f/s in YC_rC_b 4:2:0 format ((1280 x 720 x 1.5) x 8 bits x 25 f/s). Since our DSP evaluation board has not yet a frame grabber interface, a personal computer (PC) linked to a Universal Serial Bus (USB) HD webcam is used as a preliminary step to send the raw captured frames to the DSP. Our DSP board and the PC support both a Gigabit Ethernet interface (1000 Mbits/s) which allows a real-time data transfer between them. The PC could be thereafter replaced by another embedded platform including camera interface and a Gigabit Ethernet communication peripheral such BeagleBoard-xM [20] or raspberry Pi [21] platform.

As the C6678 DSP includes eight cores, the first core "core0" is considered as master. It is devoted to establish TCP/IP (transmission Control Protocol / Internet Protocol) connection with the client (PC: camera interface) exploiting Network Developer's Kit library [22]. It firstly receives the GOPs sent by the camera interface and saves them into the external memory which is a shared memory for all the DSP cores. Then, the 7 remaining DSP cores are considered as slaves and they are used to encode the 7 received GOPs. Once they finish encoding, core0 sends the bitstream of all encoded frames to the PC in order to be stored in a file or decoded later.

As shown in Fig. 4, for each slave core (1 to 7), a memory section is reserved including the GOP current frames (img1 to imgGOPsize), the reconstructed frame (RECT) and finally the bitstream buffers where the bitstream of each frame from the GOP will be saved.

In the internal memory of core0, a TCP server program is loaded to establish Gigabit Ethernet connection between the DSP and the PC. The H264/AVC program is loaded into each internal memory of the 7 remaining cores. The local variables used during the encoding such as predicted MB buffers, transform and quantification matrixes, and best predicted modes are also allocated into the internal memory of each core to avoid data overlaps among different cores and reduce memory bottleneck situation.

A C/C++ project is implemented on the PC side in order to capture raw frames from the HD camera using OpenCv library [23]. This library allows also converting the captured frames from RGB to YC_rC_b 4:2:0 format.

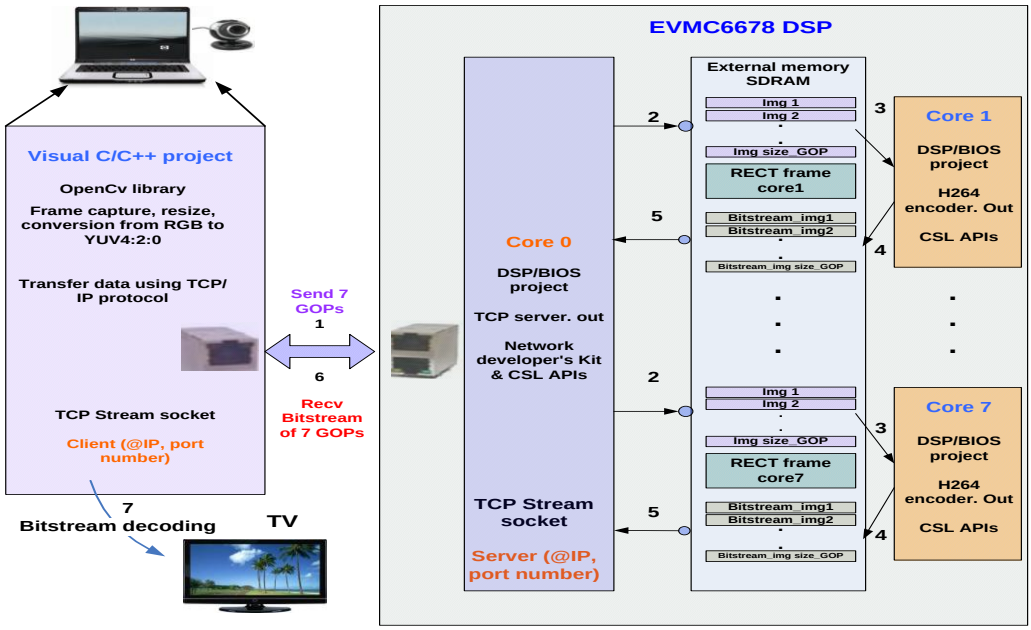

Fig. 4. Video encoding demo using the classic GOP Level parallelism approach

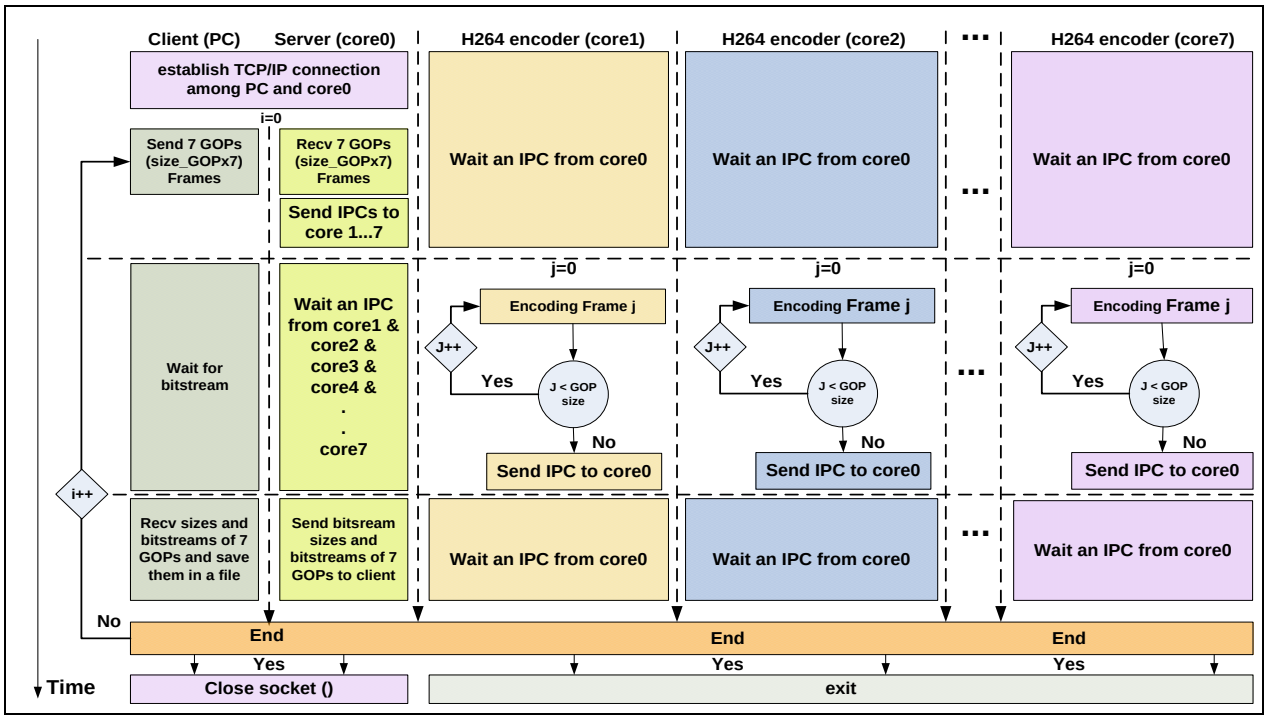

Fig. 5. The chronological steps of the classic GOP Level Parallelism approach on amulticore DSP TMS320C6678

Encoding steps using the classic GOP level parallelism approach on 7 DSP cores are described in Fig .5 and detailed as follows:

- TCP/IP connection is firstly established between the PC and the DSP (core0). The PC sends thereafter 7 GOPs to core0 which will receive them and load each GOP into its appropriate buffers as shown in Fig. 4. If only one core is used for encoding, only one GOP is sent to core0 etc.
- Once core0 finishes receiving the 7 GOPs, it sends 7 inter processor communication interruptions (IPC) to core1-core7, which are waiting for an interruption from core0, to notify them that the current frames are already in the SDRAM memory and encoding process can be started.
- As no dependencies exist among GOPs, the 7 DSP cores start the encoding process simultaneously. Each core processes its appropriate GOP frame by frame until finishing the entire GOP. For each encoded frame, a bitstream is generated and saved into its appropriate buffer in the memory section reserved for each core.
- Once the entire GOP is encoded, each core sends an IPC interruption to core0; which is in a wait state; to notify it that the bitstreams are ready to be transferred to the PC.
- When receiving the 7 IPCs from core1-core7, core0 sends the bitstream of the 7 GOPs to the PC via the Gigabit Ethernet link. Core0 sends at first the bitstream of core1 then the bitstream of core2 and finishes by the bitstream of core7 to respect the final bitstream order when it will be received by the PC and saved in a file. Before sending the bitstream data, core0 sends also the bitstream size of each frame to inform the PC about the data size which it should receive. When core0 transfers the bitstreams to the PC, the remaining cores are in a wait state for the next GOPs.
- After receiving the bitstreams and saving them in a file, the PC captures another 7 GOPs and sends them to core0. The same work thereby will be reproduced until finishing the entire video sequence.

IV.3. Cache coherency

In order to overcome the cache coherence problem [6] among DSP cores which process (read, write) the same data in a shared memory as shown in Fig. 6, TI provides the MCSDK Kit [24] which includes several libraries as the CSL (Chip Support Library) which includes also different API commands such as:

- CACHE_wbL2: it writes back the cached data from the cache memory of coreA to its location in the shared memory which allows coreB processing the updated data.
- CACHE_invL2: it invalidates the cache lines in the cache memory and forces coreA to read data from the original location in the shared memory in order to use the updated data which was processed by coreB.

In our implementation, after reading the captured frames from the PC, core0 should write back the cached data (current frames) to their locations in the external memory to be encoded later by the remaining cores. In the other side, core1-core7 should invalidate the current frames addresses in the cache memory before starting

encoding. This allows encoding the updated data written by core0 and not the old data existed in their cache memories. Moreover, once the encoding process is finishes, core1-core7 should write back the bitstreams from their cache memories to the external memory and similarly core0 should invalidate the bitstreams in its cache memory in order to send the new values to the PC.

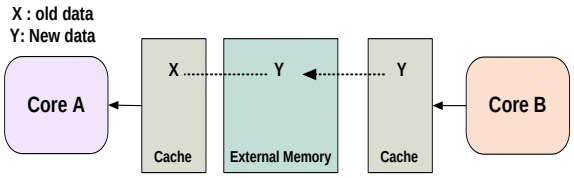

Fig. 6. Cache coherence problem

IV.4. Experimental results

The H264/AVC encoder is implemented on the TMS320C6678 multicore DSP using the LETI laboratory's software [25] [26]. This software is an optimized version of the Joint Model (JM) software. Experiments are performed on the most commonly used video test sequences in HD 720p (1280x720) resolution using different QP (Quantification Parameter) values. These sequences are a raw data in YC_rC_b 4:2:0 format recommended by the Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG organizations [27]. Encoding parameters are described in TABLE I.

TABLE I ENCODING PARAMETERS

Video resolution	1280x720
Intra period (GOP size)	8
Search range	16
QP values	30, 37
Frames to be encoded	280
Error metric for mode decision	Sum of Absolute Difference
Entropy encoder	CAVLC
Rate control	off
Nb of reference Frames	1
Optimizations	Fast intra/inter mode decision algorithms [28] [29]

For performance evaluation, encoding speed is computed with different number of slave cores and can be presented by the following equation:

$$\text{encoding speed (f/s)} = \frac{\text{DSP frequency}}{\text{Number of clock cycles}} \text{x number of frames} \quad (2)$$

Number of clock cycles indirectly represents the required time to encode a number of frames without considering communication overhead (receiving GOPs and sending bitstreams). This time is computed between sending the last IPC interruption by core0 to the last core to trigger the encoding process and the reception of the last IPC interruption from the last core which has finished the encoding process.

Table II and III show respectively the encoding speeds of the classic GOP implementation for HD video sequences with different number of cores using QP=30 and QP=37.

TABLE II ENCODING SPEEDS FOR HD720P RESOLUTION USING THE CLASSIC GOP LEVEL PARALLELISM APPROACH WITH QP=30

Video	Encoding speed using one core (f/s)	Encoding speed using 3 cores (f/s)	Encoding speed using 5 cores (f/s)	Encoding speed using 7 cores (f/s)
Shields	4.26	11.93	19.91	28.11
Parkjoy	4.91	14.22	22.52	32.09
Parkrun	4.05	11.96	19.83	27.96
sunflower	4.29	12.14	21.08	28.79
Crowdrun	3.98	11.71	18.58	25.92
Birds	4.80	13.70	23.61	32.46
Mob_cal	4.11	12.09	20.16	28.26
Stokholm	3.97	11.72	19.60	27.31
Average (f/s)	**4.29**	**12.43**	**20.66**	**28.86**

TABLE III ENCODING SPEEDS FOR HD720P RESOLUTION USING THE CLASSIC GOP LEVEL PARALLELISM APPROACH WITH QP=37

Video	Encoding speed using one core (f/s)	Encoding speed using 3 cores (f/s)	Encoding speed using 5 cores (f/s)	Encoding speed using 7 cores (f/s)
Shields	4.53	12.14	20.13	28.32
Parkjoy	5.02	14.38	22.73	32.23
Parkrun	4.36	12.26	19.96	28.19
sunflower	4.42	12.31	21.34	28.93
Crowdrun	4.12	11.83	18.78	26.24
Birds	4.96	13.84	23.86	32.72
Mob_cal	4.46	12.26	20.31	28.48
Stokholm	4.21	11.91	19.76	27.56
Average (f/s)	**4.51**	**12.61**	**20.87**	**29.08**

Experimental results show that using 7 DSP cores allows surpassing the real-time constraint 25 f/s. Encoding speed is increased from 4.29 f/s on a single core to 28.86 f/s in average on 7 cores and can reach 29 f/s with QP=37. The encoding speed is enhanced when the QP value is increased. This returns to our fast intra mode decision algorithm which is applied in the LETI's software encoder [28]. Our multicore implementation achieves an important encoding speedup of 6.73 in average with QP=30 and 6.44 with QP=37. These speedups are very close to the theoretical value (7). This tiny drop in speedup factor firstly returns to the required inter-communications among core0 and core1-core7 such as write-backs and cached data invalidations. Secondly, it returns to the impossibility of simultaneous access to SDRAM memory by the fully cores to read and write

data. Our parallel approach allows improving the encoding speed and performing a real-time HD video encoding. Our parallel implementation does not induce video quality degradation in terms of PSNR or bit-rate increase compared to a single core implementation.

V. Enhanced GOP Level Parallelism approach: hiding communication overhead

V.1. Implementation strategy

As we have mentioned earlier, the majority of published works do not consider data transfer time in their computations. When performing a real-time video coding application including video capture, frames encoding, and bitstream saving it in a file, communication overhead will be imposed and should be taken into account. As shown in Fig. 5, a lot of waiting time is noticed with the classic GOP implementation. Communication overhead is not optimized. This significantly reduces our multicore implementation efficiency. In fact, core1-core7 should wait the reception of 7 GOPs to trigger encoding process whereas encoding could be started as soon as the reception of the first frame by core0. Moreover, core0 remains in a wait state until core1-core7 finish the encoding process. However, this time could be exploited to prepare in advance the next 7 GOPs. Consequently, core1-core7 can immediately start encoding the next GOPs without waiting core0 finishing bitstream transferring and receiving the next 7 GOPs.

To enhance the classic GOP level parallelism implementation, hiding communication overhead technique is presented. Our optimization is based on two strategies as shown in Fig. 7: First, using the ping-pong buffers technique on the DSP side in order to overlap GOPs encoding process with reading and writing GOPs processes. Second, a multi-threading approach is used on the PC side. Thus, three threads are created to handle reading raw frames, sending them to DSP via Ethernet, receiving bitstreams from DSP and saving them in a file. On the DSP side, for each slave core, a ping-pong GOP buffer is allocated for both the current frames and the generated bitstreams. A single buffer is allocated for the reconstructed frame since it will not be transferred.

Consequently, one buffer for the reconstructed frame, 2* GOP size buffers for the current frames and 2* GOP size buffers also for the bitstreams are allocated in the memory section of each slave core in SDRAM memory.

Our implementation strategy is described in Fig. 8 and consists of the following steps:

- Thread1 captures the first frame from a camera or a file and sends it to core0 which will save it into the ping buffer SRC[0][0] of core1. Then, Core0 notifies core1 by sending an IPC interruption to start encoding its first current frame.
- When receiving an IPC interruption from core0, core1 starts encoding the first frame of its GOP. At the same time, thread1 continues reading the next frames of the first GOP and sending them to core0 which will save them into the ping buffers of core1 SRC1[0][i] (i=1 to GOP size-1).

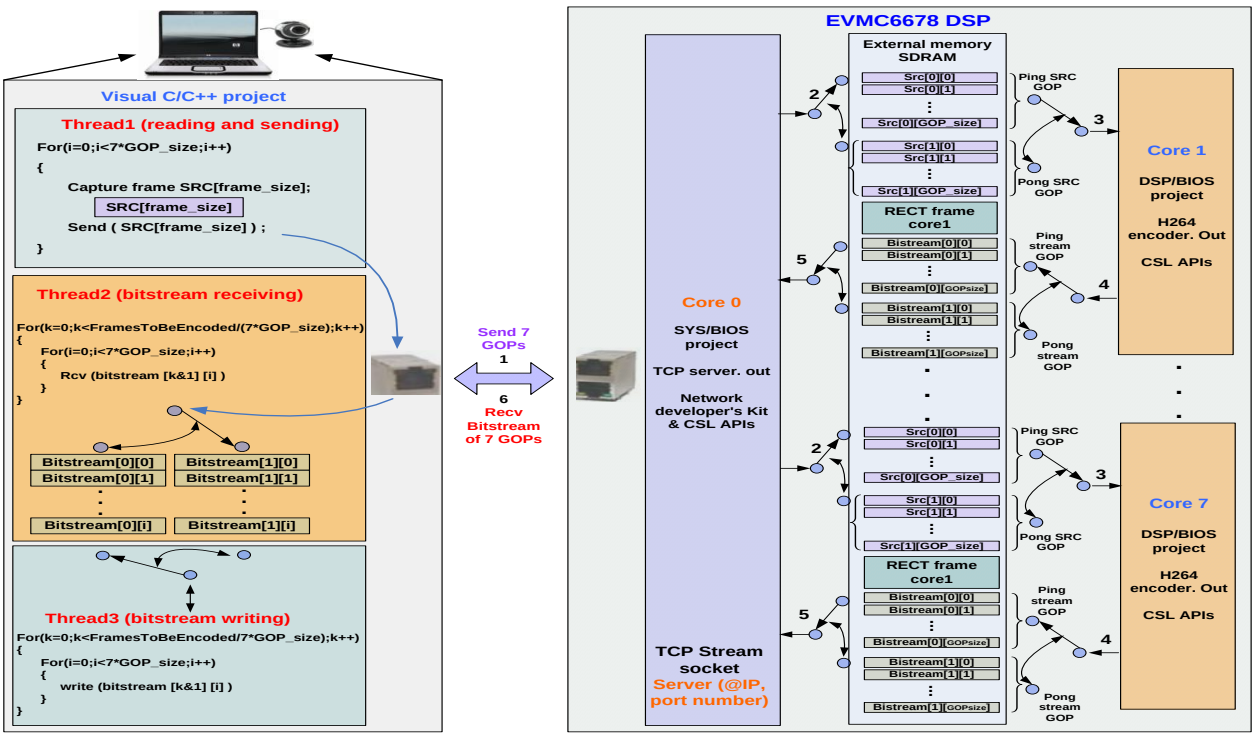

Fig. 7. Description platform using the enhanced GOP Level Parallelism approach

- When finishing reading and sending the first GOP, thread1 starts reading the second GOP and sends it to core0 which will store it into the ping buffers of core2 SRC2[0][i]. Similarly to the first GOP, when receiving the first frame of the second GOP, core0 sends an IPC interruption to core2 to notify it that it can start encoding the first frame of its GOP. This step is repeated until finishing the reception of the 7 GOPs. Thus, each core starts the encoding process immediately after receiving the first frame of the corresponding GOP without waiting the reception of all the frames.
- During encoding the first ping GOPs by core1-core7, thread1 sends the next 7 GOPs to core0 which will store them into the pong buffers SRC[1][i] of each core (i=0 to GOP size-1). As encoding process takes more time than reading process, communication delays are hidden and they do not contribute to the parallel run-time.
- Once core *i* finishes the encoding process and saves the bitstream into the ping buffers Bistream[0][i], it sends an IPC interruption to core0 to notify it that it can forward its bitstream to the PC. Then, this core starts encoding its pong GOP, already received and stored into the pong buffers SRC[1][i] without any wait. Consequently, the new generated bitstream will be stored into the pong buffers Bistream[1][i] to not overwrite data stored in the ping buffers Bistream[0][i] which are being transferred by core0 to the PC.
- During encoding the pong GOPs, core0 sends the ping bitstreams (Bistream[0][i]) to the PC starting with those of core1 and finishing with the bitstreams of core7 in order to be saved in a chronological order. Thus, thread2 receives these ping bitstreams and stores them into the ping buffers Bitstream[0][i]. Finally, thread3 writes them in a file and at the same time thread1 sends the next 7 GOPs to core0 which will store them into the ping buffers SRC[0][i] of each core. With this strategy, the ping bitstreams writing, the pong SRC GOPs encoding and the next 7 ping GOPs capturing and sending are practically processed in parallel.
- The encoding process is then reproduced in a reverse order for SRC frames and bitstreams through ping-pong buffers.

When looking at Fig. 8, no significant delays are noted. Core1-core7 process their corresponding data without any waiting time.

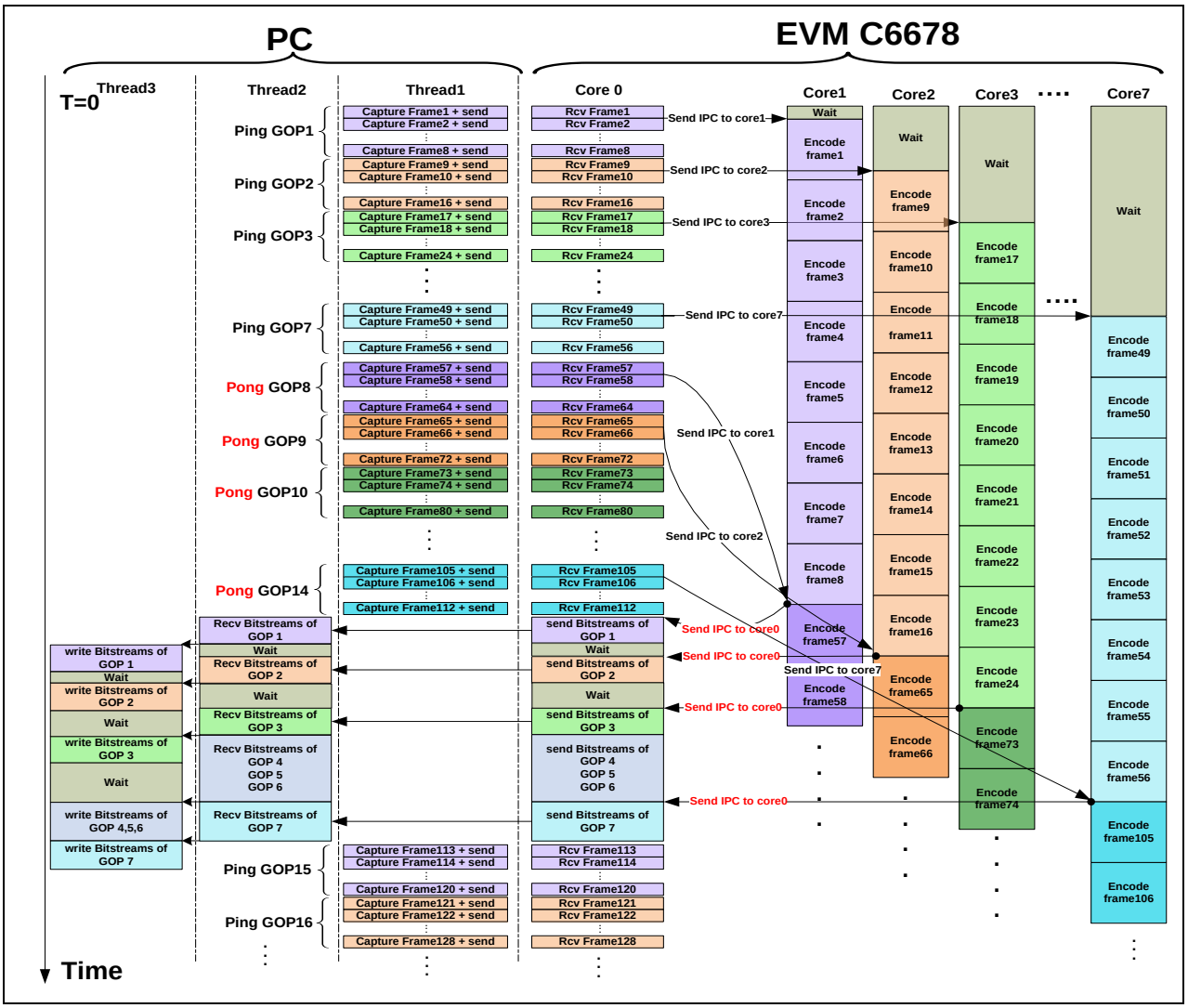

Fig. 8. The chronological steps of the Enhanced GOP Level Parallelism on the multicore DSP TMS320C6678

V.2. Experimental results for the Enhanced GOP implementation

To evaluate our enhancement, communication overhead is considered in the encoding time computation. Frames capture, transferring them to DSP, receiving them by core0 and loading them to DSP memory are taken into account in our calculation. Table IV shows the encoding speeds for both: the classic GOP implementation (without including data transfer times: only encoding time) and the enhanced GOP implementation for HD resolution with QP=30. The same encoding parameters, used in the first implementation, are reused for the enhanced approach.

TABLE IV ENCODING SPEED FOR HD RESOLUTION USING THE ENHANCED GOP LEVEL PARALLELISM APPROACH

HD Video sequences	Encoding speed on 7 DSP cores: Classic GOP imlementation (data transfer is not included) (f/s)	Encoding speed on 7 DSP cores: Enhanced GOP imlementation (data transfer is included) (f/s)
Shields	28.11	27.38
Parkjoy	32.09	31.14
Parkrun	27.96	27.25
sunflower	28.79	27.86
Crowdrun	25.92	25.27
Birds	32.46	31.58
Mob_cal	28.26	27.48
Stokholm	27.31	26.43
Average (f/s)	**28.86**	**28.04**

As shown in Table IV, a non-significant drop in encoding speed is induced. This clearly affirms that our proposed data transfer scheduling technique efficiently hides communication overhead. The obtained encoding speeds for the enhanced implementation are very close to those of the classic GOP implementation. In fact, data transfer time is only noticed at the first GOPs. After that, these transfers are overlapped with the encoding process.

Experiments show that our enhanced implementation allows surpassing the real-time compliant (25 f/s) by reaching up to 28 f/s in average for HD video sequences. An important speedup factor of 6.7 is obtained on 7 DSP cores. Regarding rate distortion, our multicore implementation does not induce visual quality degradation or bitrate increase compared to a unique core implementation.

It may be noted that several factors are contributed to achieve this performance despite that encoding steps, detailed above, transmit the idea that there is always a simultaneous accesses to the external memory by the different cores which may cause a significant latency.

First, our encoding implementation is based on a "MB row level architecture" [26]; so each core copies a MB row from the current frame in the external memory into a current MB row buffer in the internal L2 memory. The encoding process will be thereafter performed by the CPU between the L1 and L2 level memories which reduces the external memory bottleneck situation. Secondly, 128 kbytes of L2 memory are configured as cache for each core. Thus, access to a memory location triggers a prefetch of a line of memory locations into the cache memory. This allows reducing the cache misses; so accelerating encoding run-time. Reconstructed MBs row and bitstream are not directly copied into the external memory after their processing but they remain in the cache memory which reduces the external memory access. Third, in addition to eight processing units for each core which allow performing eight instructions per cycle, code composer studio IDE (Integrated Development Environment for DSP programming) allows generating an optimized assembler code that exploits the maximum of pipeline. Thus, the different cores may do not perform the same load instruction from the external memory at the same time, a core *i* can perform prefetch instructions, other core can perform load instruction and another one can execute ADD instructions for example etc. Moreover, our enhanced implementation is a pipelined design; there is a timing delay between the different cores. So reading current MB rows and writing bitstreams are not necessarily performed at the same time by all cores. Furthermore, the C6678 external memory is characterized by large bandwidth. In fact, it is a 64-bit DDR3 SDRAM operating at 1333 MHz with up to 10664 Mbyte/s of throughput ((64bits)/(8bits/byte)*1333 MHz). Several tests show that this bandwidth is enough to support multiple DSP cores accessing the DDR3 memory simultaneously [30].

To more evaluate our enhanced approach, the total required time to perform the entire video encoding chain (frames capturing, DSP encoding, and bitstream in a file) is computed for the both implementations: the classic and the enhanced one.

Table V shows the computed time of the fully encoding chain for these two implementations. The number of HD encoded frames is 1200. This time depends on the PC performance (reading/writing data from/in a file), DSP performance (H264/AVC encoding process) and the data transmission bandwidth between the PC and the DSP (sending/receiving data using TCP/IP Ethernet protocol). Our DSP frequency is 1 GHz. The used PC is Intel i3 running at 2.33 GHz and including a Gigabit Ethernet communication interface.

TABLE V THE TOTAL TIME OF THE ENTIRE ENCODING CHAIN

HD video sequences	The total time for the classic GOP implementation (second)	The total time for the enhanced GOP implementation (second)	Saving time (%)
Shields	241	148	38.58%
Parkjoy	210	132	37.14%
Parkrun	256	162	36.71%
sunflower	237	143	39.66%
Crowdrun	231	147	36.36%
Birds	242	151	37.60%
Mob_cal	216	136	37.03%
Stokholm	234	155	33.76%
Average time (s)	**233**	**147**	**36.90%**

Experimental results show that our enhancements allow saving up to 36% of the total time for HD video encoding chain. Exploiting multi-threading algorithm with ping-pong buffers technique significantly reduces the processing time by overlapping the encoding process with that of data transferring.

TABLE VI COMPARISON WITH PREVIOUS WORKS

approach	Our approach	[2]	[10]	[11]	[14]	[15]	[17]
Partitioning method	GOP	Task	GOP	MB/Frame	slice	MB region partition (MBRP)	Task
platform	Multicore DSP TMS320C6678 (7 cores for encoding)	167-core asynchronous array of simple processors	3 Microblaze soft cores based on XILINX FPGA	Pentium 4 processor running at 2.8 GHz	quad TMS320C6201 DSP system	PC with a P4 1.7GHz processor 4 cores	NVIDEA's GPU using CUDA with 448 cores
Reference software and encoding parameters	LETI's H264 codec, baseline profile, ME algorithm is LDPS, search range=16, Number of reference frame=1, R-D optimization is not used, entropy coding is CAVLC.	JM baseline profile, search range=3, ME algorithm is Diamond Search, Number of reference frame=1, entropy coding is CAVLC.	AVS reference code RM5.2, ME algorithm is full search, entropy coding is CAVLC.	JM9.0, one reference frame for MV, search range=10, R-D optimization is used, entropy coding is CAVLC.	H263/MPEG4 baseline profile, search range=16, ME algorithm is diamond search, entropy coding is VLC.	JM 10.2 baseline profile, ME algorithm is the Full search, Number of reference frame=1, R-D optimization is used, entropy coding is CAVLC.	X264 codec, search range=32, ME algorithm is MRMW, Number of reference frame=1, entropy coding is CAVLC.
Encoding speed (f/s)	28 f/s for HD	21 f/s for VGA (640 x 480)	3 f/s for QCIF	0.58 f/s for CIF	30 f/s only for CIF resolution	0.6 f/s for CIF and 0.15 f/s for SD	30 f/s for HD720p
Distortion PSNR/bitrate	No	yes	No	No	yes	No	Yes

For low and medium video resolutions such as CIF(352x288), VGA (640x480) and SD (720x480), real-time is achieved on less than 7 cores which allows exploiting the remaining cores to perform other tasks such biometric recognition, access control, objects detection and surveillance application etc. This will give an important advantage for our multicore DSP if integrated into a smart system.

For more performance evaluation, our solution is compared to previous works which have been performed on several platforms and applied different parallelism methods. As shown in Table VI, several implementations have not satisfied the real-time constraint. In fact, JM software is not an optimized algorithm which makes it hard to reach a real-time encoding performance. Some works have achieved the real-time compliant for low resolution but not yet for higher resolutions. GPU implementation [17] allows performing a real-time HD video encoding thanks to the great number of processing cores. However, this proposed scheme induces some rate distortion (PSNR degradation and bitrate increase). Finally, we can note that our implementation has ensured a good encoding scalability without inducing any rate distortion compared to a single core implementation.

In addition to the performance evaluation with previous works, our H264/AVC encoder implementation based on LETI's codec is also compared to the JM 18.6 reference software. Encoding performance is evaluated in terms of:

- ΔPSNR (dB): it presents the visual quality degradation in terms of PSNR when using our encoder compared to the JM reference software.
- ΔBitrate (%): it presents the percentage increase in bitrate when using our encoder compared to the JM reference software.
- Encoding speed (f/s): it depends on the CPU frequency and the encoder computational complexity.

These above criteria are presented by the following equations:

$$\Delta\text{Bits}(\%) = \frac{\text{Bitrate}(\text{LETI's codec}) - \text{Bitrate}(\text{reference JM})}{\text{Bitrate}(\text{reference JM})} * 100 \quad (3)$$

$$\Delta\text{PSNR (dB)} = \text{PSNR}(\text{LETI's codec}) - \text{PSNR}(\text{JM}) \quad (4)$$

As we noted above, LETI's codec is an optimized version of the JM software. We have applied various optimizations for the different modules (mode decision, motion estimation, ICT transform, and de-blocking filter) to reduce the computational complexity of this encoder and achieve a compatible DSP-based solution. Furthermore, some functions have been programmed in assembler language to efficiently exploit the internal resources of our DSP.

The JM18.6 encoder software is processed on an Intel core2 Quad CPU running at 2.33 GHz. Our LETI's encoder is evaluated on the multicore Keystone DSP TMS320C6678 running at 1 GHz each core. Simulation parameters are detailed in Table VII.

Table VIII shows the encoding performances in terms of the three cited criteria for the both implementations. Experimental results show that the JM reference software ensures better encoding performances in terms of PSNR and bitrate compared to our encoder. In fact, our H264/AVC encoder induces PSNR degradation by 1 dB in average and an increase in bitrate by 3% compared to the JM18.6 reference software.

TABLE VII ENCODING PARAMETERS USED FOR THE JM 18.6 AND THE LETI'S CODEC

	JM 18.6	LETI's Codec
Target platform	Intel core2 Quad CPU Q8200 running at 2.33 GHz each CPU	Multicore DSP TMS320C6678 running at 1GHz each core
Video resolution	HD (1280x720)	HD (1280x720)
Quantification parameter (QP)	30	30
Frame rate	25	25
Intra period	8	8
Motion estimation algorithm	EPZS	LDPS
Subpixel Motion Estimation	on	off
Error metric	SAD	SAD
Number of reference frame	1	1
Entropy coding method	CAVLC	CAVLC
Rate control	off	off
Rate Distortion Optimized	disabled	disabled
Function optimizations	Non	Fast intra and inter prediction algorithms, Fast mode decision algorithm, optimized filtering module
Software optimizations	Visual studio optimizations : Maximize speed, favor fast code, Multi-threaded Debug …etc.	Code composer optimizations : intrinsic functions, using assembler language for some modules+ Enhanced GOP Parallelism on 7 cores

TABLE VIII ENCODING PERFORMANCES FOR THE JM 18.6 AND THE LETI'S CODEC

HD video sequences	PSNR (dB) (JM)	ΔPSNR (dB)	Bitrate JM (Kbit/s)	ΔBitrate (%)	Encoding time for JM (f/s)	Encoding time for our encoder (f/s)
stockholm	34,68	-0,6	3934	+2,53	1,09	27,31
sunflower	40,63	-0,98	2365	+2,69	1,10	28,79
mob_cal	33,67	-0,98	7136	+2,68	1,11	27,48
crowdrun	34,65	-1,5	11309	+3,16	1,06	25,92
shields	34,88	-0,9	5161	+2,83	1,10	27,38

Regarding encoding speed, we can note that our encoder is more optimized and faster than the JM reference software. Our multicore DSP implementation allows performing a real-time HD video encoding by reaching up to 28 f/s whereas, JM reference software is not able to meet the real-time compliant. This returns to the various optimizations applied to reduce the computational complexity and accelerate the encoding process.

V.3. Power consumption estimation

To estimate the power consumption of our H264/AVC encoder implementation on the TMS320C6678 DSP, we have adopted the TI's spreadsheet [31] as shown in Fig. 9. It is an excel file which includes configurable parameters that allow estimating the power consumption based on configured usage parameters.

These parameters are presented as follows:

- Frequency: specifies the frequency of the DSP core or the frequency of the external interface as DDR3.
- Modes: selects the peripheral-specific configuration mode.
- Status: indicates whether a peripheral is Enabled (used) or Disabled (unused).
- % Utilization: specifies the percentage of the time the module spends doing something useful, versus being unused or idle. It includes the % Signal Processing (SP) Utilization, % Control Code (CC) Utilization, and % Idle Utilization.
- %SP: represents scenarios with high levels of DSP activity. This corresponds to all 8 instructions fetched by the DSP executed in parallel each DSP clock cycle. Thus all 8 functional units are active every cycle.
- %CC: represents scenarios with low levels of DSP activity. It represents execution of approximately 2 functional units every clock cycle.
- % Write: represents the relative amount of time the module is transmitting versus receiving.
- Bits: specifies the number of data bits to be used in a selectable-width interface.
- Lane: specifies the number of lanes used by that interface.
- % Switching: specifies the probability that any one data bit on the relative data bus will change state from one cycle to the next.

More details about these parameters are presented in the reference of Power Consumption Summary for KeyStone C66x Devices [32].

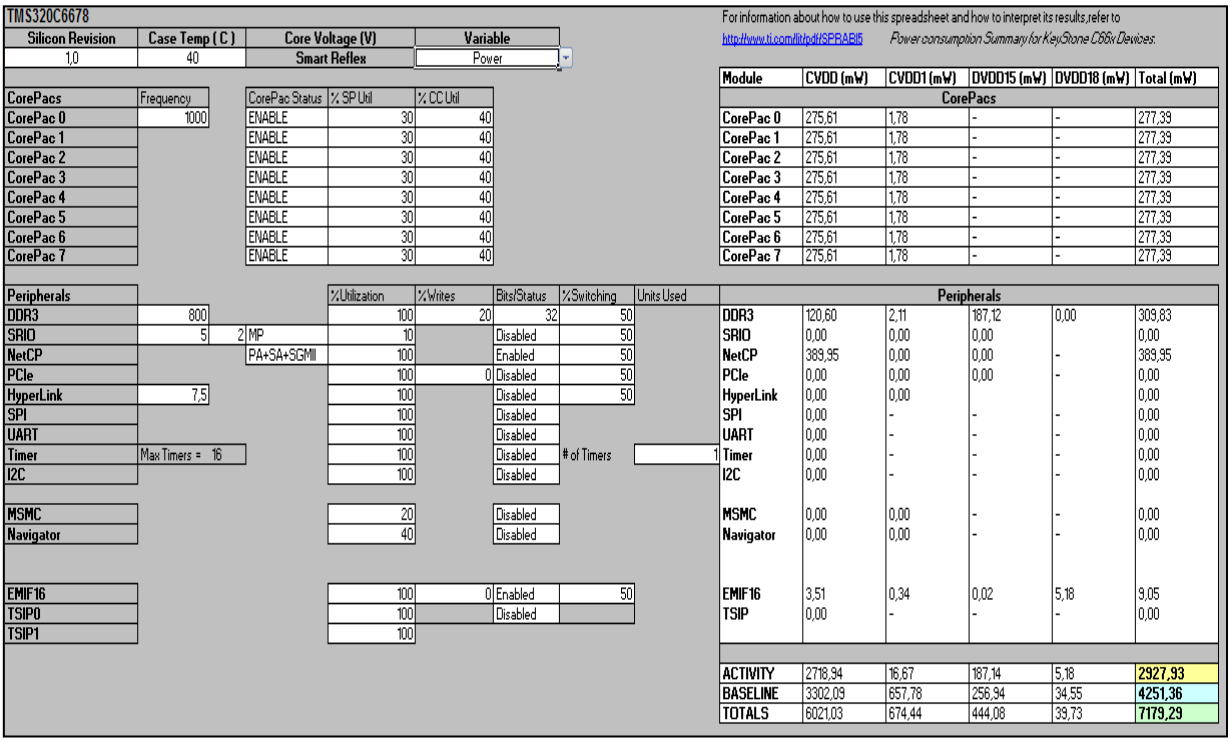

Fig. 9. Estimation of power consumption using TI's spreadsheet

In our estimation, we specify 30% and 40% respectively for the %SP and the %CC utilizations. This specification presents a more realistic scenario for a very signal processing intensive code [33]. In fact, a very few kernels achieve 8 operations per cycle.

As external interfaces, we enabled only the DDR3, EMIF16, and the NetCP and we supposed working at 40° C of temperature. The estimated power consumption is equal to 7.2 W (watt) as shown in Fig. 9. This consumption value is considered non-significant compared to GPU platforms or GPP processors (General Purpose Processors) [29].

VI. Conclusion

In this paper, an optimized H264/AVC HD video encoder implementation on a multicore DSP TMS320C6678 was presented. GOP Level parallelism approach was applied to accelerate encoding speed. Exploiting the ping-pong buffers technique with a multi-threading algorithm allows hiding communication overhead and efficiently enhances the encoding performance. Experimental results on 7 DSP cores running each at 1 GHz proved that our enhanced implementation has met the real-time encoding compliant. The achieved encoding speed is up to 28 f/s in average for HD resolution. Our parallel implementation allowed accelerating the encoding process by a factor of 6.7 without inducing a PSNR drop or bitrate increase compared to a single core implementation. Compared to the JM18.6 reference software, our LETI's optimized software induced visual quality degradation by 1 db in terms of PSNR and 3% of bitrate increase. This rate distortion is acceptable when looking at the important encoding speedup and the HD real-time processing. The proposed scheduling technique for hiding communication overhead allowed saving up to 36% of the fully encoding chain time. Power consumption of our multicore implementation was estimated to 7.2 W which is considered non-significant compared to GPU's or GPP's power consumption. As perspectives, we will move to implement the new video coding standard HEVC (High Efficiency Video Coding) on our multicore DSP TMS320C6678. The same proposed technique could be reapplied for this recent encoder. In fact, HEVC almost adopts the same hierarchical data video structure of H264/AVC encoder (GOPs, frames, slices, MB), and practically, the same dependencies in H264AVC encoder exist among HEVC data units.

Acknowledgements

This work is fruit of cooperation between Sfax National School of Engineers and ESIEE PARIS Engineering School. It is sponsored by the French ministries of Foreign Affairs and Tunisian ministry for Higher Education and Scientific Research in the context of Hubert Curien Partnership (PHC UTIQUE) under the CMCU project number 12G1108.

References

[1] Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG, "Advanced video coding for generic audiovisual services", Avril 2013. Online available: http://www.itu.int/ITU-T/recommendations/rec.aspx?rec=11830&lang=en

[2] Zhibin Xiao, Stephen Le and Bevan Baas," A Fine-grained Parallel Implementation of a H.264/AVC Encoder on a 167-processor Computational Platform," ACSSC 2011 – Pacific Grove, CA, 2011.

[3] Ming-Jiang Yang; Jo-Yew Tham; Rahardja, S.; Da-Jun Wu, "Real-time H.264 encoder implementation on a low-power digital signal processor," Multimedia and Expo, 2009. ICME 2009. IEEE International Conference on , vol., no., pp.1150,1153, June 28 2009-July 3 2009

[4] Seongmin Jo, Song Hyun Jo, Yong Ho Song, "Exploring parallelization techniques based on OpenMP in H.264/AVC encoder for embedded multi-core processor," Journal of Systems Architecture, Volume 58, Issue 9, October 2012, Pages 339-353.

[5] Hajer Krichene Zrida, Ahmed C. Ammari, Abderrazek Jemai, Mohamed Abid," High Level Optimized Parallel Specificationof a H.264/AVC Video Encoder," International Journal of Computing & Information Sciences Vol. 9, No. 1, Pages 34 – 46 April 2011.

[6] TMS320C6000 DSP Cache User's Guide, online available: http://www.ti.com/lit/ug/spru656a/spru656a.pdf

[7] S.Sankaraiah, H.S.Lam, C.Eswaran and Junaidi Abdullah, "GOP Level Parallelism on H.264 Video Encoder for Multicore Architecture," International Conference on Circuits, System and SimulationIPCSIT vol.7 IACSIT Press, Singapore 2011.

[8] S. Sankaraiah ,Lam Hai Shuan, C. Eswaran and Junaidi Abdullah , "Performance Optimization of Video Coding Process on Multi-Core Platform Using Gop Level Parallelism" International Journal of Parallel Programming,ISSN:1573-7640, DOI 10.1007/s10766-013-0267-4, September2013.

[9] Rodriguez, A.; Gonzalez, A.; Malumbres, M.P., "Hierarchical Parallelization of an H.264/AVC Video Encoder," Parallel Computing in Electrical Engineering, 2006. PAR ELEC 2006. International Symposium on , vol., no., pp.363,368, 13-17 Sept. 2006

[10] Fang Ji; Xing-yuan Li; Chang-long Yang, "An Algorithm Based on AVS Encoding on FPGA Multi-Core Pipeline," Computational and Information Sciences (ICCIS), 2013 Fifth International Conference on , vol., no., pp.1521,1524, 21-23 June 2013.

[11] Zhuo Zhao; Ping Liang, "A Highly Efficient Parallel Algorithm for H.264 Video Encoder," Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on , vol.5, no., pp.V,V, 14-19 May 2006.

[12] H264/AVC software Joint Model JM, online available: http://iphome.hhi.de/suehring/tml/download/old_jm/

[13] Yen-Kuang Chen; Tian, X.; Steven Ge; Girkar, M., "Towards efficient multi-level threading of H.264 encoder on Intel hyper-threading architectures," Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International , vol., no., pp.63,, 26-30 April 2004.

[14] Olli Lehtoranta, Timo Hämäläinen, Ville Lappalainen, Juha Mustonen, "Parallel implementation of video encoder on quad DSPsystem,"Microprocessors and Microsystems, Volume 26, Issue 1, Pages 1-15, 25 February 2002.

[15] Sun, S.; Wang, D. & Chen, S. Perrott, R.; Chapman, B.; Subhlok, J.; Mello, R. & Yang, L. (Eds.), "A Highly Efficient Parallel Algorithm for H.264 Encoder Based on Macro-Block Region Partition," High Performance Computing and Communications, Springer Berlin Heidelberg, 2007, 4782, 577-585

[16] Shenggang Chen; Shuming Chen; Huitao Gu; Hu Chen; Yaming Yin; Xiaowen Chen; Shuwei Sun; Sheng Liu; Yaohua Wang, "Mapping of H.264/AVC Encoder on a Hierarchical Chip Multicore DSP Platform," High Performance Computing and Communications (HPCC), 2010 12th IEEE International Conference on , vol., no., pp.465,470, 1-3 Sept. 2010

[17] Huayou Su, Mei Wen, Nan Wu, Ju Ren, and Chunyuan Zhang, "Efficient Parallel Video Processing Techniques on GPU: From Framework to Implementation," The Scientific World Journal, vol. 2014, Article ID 716020, 19 pages, 2014.

[18] António Rodrigues, Nuno Roma, and Leonel Sousa," p264: Open Platform for Designing Parallel H.264/AVCVideo Encoders on Multi-Core Systems," NOSSDAV '10 Proceedings of the 20th international workshop on Network and operating systems support for digital audio and video Pages 81-86, Amsterdam, The Netherlands, 2010.

[19] TMS320C6678 Multicore Fixed and Floating-Point Digital Signal Processor Data Manual, Literature Number: SPRS691D April 2013, online available:http://www.mouser.com/ds/2/405/sprs691d-256638.pdf

[20] BeagleBoard-xM Rev C system Reference Manual, online available: http://beagleboard.org/static/BBxMSRM_latest.pdf

[21] Raspberry Pi reference manual, online available: http://www.raspberrypi.org/tag/raspberry-pi-user-guide/

[22] TI Network Developer's Kit (NDK) v2.21 User's Guide, online available: http://www.ti.com/lit/ug/spru523h/spru523h.pdf

[23] Open source computer vision library,online available: http://opencv.org/

[24] SYS/BIOS and Linux Multicore Software Development Kits (MCSDK) for C66x, C647x, C645x Processors, online available: http://www.ti.com/tool/bioslinuxmcsdk

[25] I. Werda, F. Kossentini, M.A.B. Ayed, and N. Masmoudi, "Analysis and Optimization of UB Video's H.264 Baseline Encoder Implementation on Texas Instruments' TMS320DM642 DSP", ;in Proc. ICIP, 2006, pp.3277-3280.

[26] N.Bahri, I.Werda, T.Grandpierre, M.Ben Ayed, N.Masmoudi, M.Akil," Optimizations for Real-Time Implementation of H264/AVC Video Encoder on DSP Processor," International Review on Computers and Software (I.RE.CO.S.), Vol. 8, n. 9, pp.2025-2035 september 2013.

[27] Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG organizations, online available:http://www.itu.int/en/ITU-T/studygroups/com16/video/Pages/jvt.aspx

[28] Nejmeddine Bahri, Imen Werda, Amine Samet, Mohamed Ali Ben Ayed and Nouri Masmoudi. Article, "Fast Intra Mode Decision Algorithm for H264/AVC HD Baseline Profile Encoder," International Journal of Computer Applications 37(6):8-13, January 2012.

[29] Imen Werda, Haithem Chaouch, Amine Samet, Mohamed Ali Ben Ayed, Nouri Masmoudi, "Optimal DSP Based Integer Motion Estimation Implementation for H.264/AVC Baseline Encoder," The International Arab Journal of Information Technology - IAJIT , vol. 7, no. 1, pp. 96-104, 2010.

[30] Throughput Performance Guide for C66x KeyStone Devices, online available:http://www.ti.com/lit/an/sprabk5a/sprabk5a.pdf

[31] C6678 power spreadsheet, online available: http://www.ti.com/lit/zip/sprm545

[32] Power Consumption Summary for KeyStone C66x Devices, online available: http://www.ti.com/lit/an/sprabi5a/sprabi5a.pdf

[33] C6678 Power spreadsheet, online available: http://e2e.ti.com/support/dsp/c6000_multi-core_dsps/f/639/t/171805.aspx