Real-time H264/AVC high definition video encoder on a multicore DSP TMS320C6678

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

Real-time H264/AVC High Definition video encoder on a Multicore DSP TMS320C6678

Nejmeddine Bahri, Nidhameddine Belhadj, Med Ali Ben Ayed, Nouri Masmoudi
National Shool of Engineers of Sfax, LETI Laboratory,
University of Sfax, Tunisia
nejmeddine.bahri@esiee.fr

Thierry Grandpierre, Mohamed Akil
ESIEE Engineering, LIGM Laboratory,
University Paris-EST, France
thierry.grandpierre@esiee.fr

***Abstract*—In this paper, the newest Texas Instrument's multicore DSP TMS320C6678 is used in order to perform a real-time H264/AVC high definition (HD) embedded video encoder. We exploit the high computing performance offered by this eight-core DSP in order to meet the real-time encoding compliant. To enhance the encoding speed, Frame Level Parallelism (FLP) approach is applied. A master core is reserved to handle data transfers to/from DSP. Multithreading algorithm combined with a ping-pong buffers technique are exploited in order to optimize the standard FLP approach and hide communication overhead. Experimental results show that our enhanced FLP implementation allows achieving real-time HD (1280x720) video encoding by reaching up to 26 f/s (frame/second) as encoding speed. Experiments show also that our parallel implementation, performed on seven C6678 DSP cores running each @ 1 GHz, allows accelerating the encoding run-time by a factor of 6,38 without inducing any quality degradation or bit-rate increase.**

I. INTRODUCTION

Nowadays, HD resolution is widely used in several multimedia video applications due to the rapid evolution of digital cameras technology. Facing the high cost of storing raw video data and transmission bandwidth limitation, video encoding with high compression performance is absolutely required. H264/AVC encoder represents one of the most efficient video standards. This encoder ensures a high encoding efficiency compared to previous standards by saving up to 50% of bit-rate while maintaining the same visual quality. However, this efficiency comes with a tremendous computational complexity which makes it hard to meet the real-time video encoding constraint (25 f/s) for HD resolution.

In order to overcome this complexity, a high performance computing capability is absolutely required. Single core processors with low CPU frequency are not able to meet real-time HD video coding. Consequently, using multicore, multithreading, and multiprocessor platforms can be a promising solution for this problem. Several works which profit from the potential parallelism in H264/AVC encoder have been presented in several papers [2]-[6]. These works are based on applying different partitioning techniques and exploited both task and data level parallelism.

In this context, we try to develop our embedded video encoder solution which is characterized by software flexibility and low power consumption. Our proposed solution should meet HD encoding requirements in terms of visual quality, real-time processing, and compression performance. Consequently, it could be included in various video applications such as smart cameras, machine vision computing, smart TV, mobile phone, robotic, traffic security, and surveillance camera etc.

To achieve this aim, the Keystone TI's multicore DSP TMS320C6678 is used in order to perform a parallel H264/AVC video encoder implementation. Frame Level Parallelism approach is selected to accelerate data processing time. Multithreading algorithm combined with ping-pong buffers technique are exploited in order to improve the encoding performance and hide communication overhead.

The rest of this paper is outlined as follows: next section details the different partitioning approaches and some parallel H264/AVC video encoder implementations. TMS320C6678 architecture overview is described in section 3. Section 4 highlights our enhanced FLP implementation on eight C66x cores and presents experimental results. Finally, section 5 concludes this paper and presents some perspectives.

II. H264/AVC ENCODER : PARTITIONING METHODS AND PARALLEL IMPLEMENTATIONS

H.264/AVC baseline encoder is a video compression standard that aims to reduce the large amount of raw video data in order to overcome the bandwidth transmission limitation and reduce the raw video storing cost. The main structure of this standard consists in performing several tasks in order to ensure an efficient encoding performance such as: intra prediction, inter prediction, integer cosine transform, quantification, filtering, and entropy coding. This standard splits a video sequence into hierarchical data structure. In fact, the sequence is divided into one or more groups of pictures (GOP). Each GOP includes one or more frames. The first frame of the GOP is an intra frame (I) and the remaining ones are predicted frames (P). Each frame can be divided into one or more slices, subdivided themselves into macroblocks (MB) and blocks.

In order to perform a parallel implementation for this encoder, two partitioning approaches can be exploited:

Task-level parallelization (TLP): it exploits the functional organization of H264/AVC encoder by regrouping the different functions in several tasks, equal to the number of processing units available on the system, and run these tasks method as in [2]. This approach ensures low encoding latency; however, it

This work is supported by the French ministry of Foreign Affairs and the Tunisian ministry for Higher Education and Scientific Research in the context of Hubert Curien Partnership (PHC UTIQUE) under the CMCU project number 12G1108.

has some drawbacks. First, we can say that it is not very suitable for H264/AVC encoder because the existence of a lot of data dependencies among tasks which requires a large amount of data transfers among processors. Second, a lot of inter-processors synchronizations are required which increases the implementation complexity. Finally, functions in H.264/AVC encoder have not the same load balance which makes it hard to uniformly map functions among processors; consequently, encoding performance is always depending on the heaviest load processor.

Data-level parallelization (DLP): it profits from the hierarchical data structure of H264/AVC encoder by simultaneously processing several data levels on multiple processing units. Some data dependencies should be respected among the different data when applying this approach.

First, no dependencies exist among different GOPs. In fact, each GOP starts with an intra frame (the current MB encoding requires only some data from its neighboring MBs in the same frame). Consequently, several GOPs can be processed in parallel. This approach is called "GOP Level Parallelism". It has been adopted by several researchers as in [3].This method ensures the best encoding speedup and characterized by low synchronization cost and no data communications among processors. However, it requires a large memory amount in order to handle all the frames and it is characterized by a high encoding latency. Consequently, it is not suitable for System on Chip platforms (SOC) and real-time video applications.

Temporal dependency is imposed by the motion estimation algorithm between successive frames of the same GOP. In fact, to determine the current MB motion vector, a block matching algorithm is performed in a restricted search window in the reference frames (previous frames already encoded) as shown in Fig .1. Accordingly, multiple frames can be encoded in parallel way once the search window in the reference frame has already been encoded. This method is called "Frame Level Parallelism" [4]. This approach provides a compromise between encoding latency and implementation efficiency. In fact, Memory amount and encoding latency is significantly reduced compared to GOP parallel implementation. In the other side, processors synchronizations are required in order to respect the temporal data dependency.

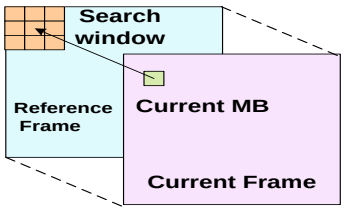

Fig. 1. Data dependency for inter prediction

Since H264/AVC standard gives the possibility to divide the frame into independent slices, several slices can be processed in parallel, and this approach is named by "Slice Level Parallelism" such as in [5]. This approach is characterized by high scalability and low encoding latency but it has a major drawback. In fact, it induces visual quality degradation in terms of PSNR and an increase in bit-rate. This returns to data dependencies that are not respected among MBs of different slices.

Finally, in the frame itself, several MBs can be encoded in parallel once its neighboring MBs have already been processed in order to respect spatial data dependencies required by the intra prediction module as shown in Fig. 2.

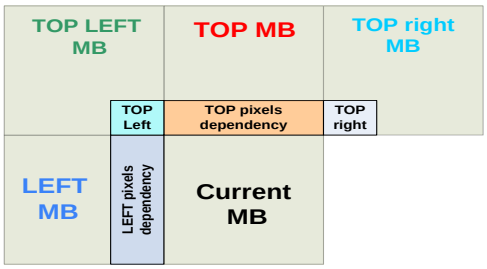

Fig. 2. spatial dependencies for the current MB

This approach is called "MB level Parallelism" [6]. It is characterized by low encoding latency and low memory requirement but in the other side, low scalability, large amount of data transfers and synchronizations among processors in addition to the non-equal load balance make MB level parallelism approach not efficient for parallel H264/AVC encoder implementation.

In the state of art, some parallel H264/AVC encoder implementations have not yet succeeded to meet the real-time compliant especially for high resolutions. Consequently, we will present in this work our strategy to perform a real-time parallel H264/AVC video coding without inducing any distortion in terms of visual quality or bit-rate. Our scheme is based on exploiting the merits of a multicore platform and applying an efficient partitioning method.

III. DSP Platform Description

New generations of DSP processors represent an attractive solution for embedded applications that require high computing performance. In fact, these DSPs are characterized by software flexibility, high performance computing, multicore architecture, low power consumption, competitive price tag, and time to market. According to these merits, we chose to implement the H264/AVC encoder on the last generation of TI's keystone multicore DSP TMS320C6678 [7] in order to get a flexible embedded encoder that allows achieving a real-time 25 f/s HD video encoding. As shown in Fig. 3, eight DSP Core Subsystems (C66x CorePacs) running each @ 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. To support applications that require a large amount of memory such as ultra HD video applications, TMS320C6678L includes a 512MB of DDR3-1333 external memory. This EVM platform (evaluation module) comes with TI's Multicore Software Development Kit (MCSDK) for SYS/BIOS Real Time Operating System (RTOS). For external communications, TMS320C6678 supports several high speed standard interfaces such as RapidIO for DSP-to-DSP communications and Gigabit Ethernet for Internet Protocol (IP) networks etc.

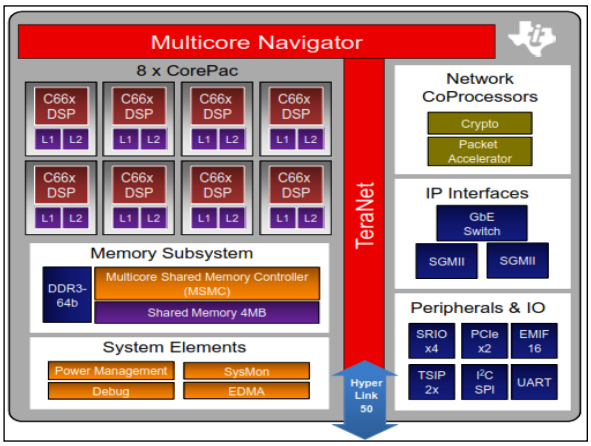

Fig. 3. Internal architecture of TMS320C6678 multi-core DSP

IV. ENHANCED FRAME LEVEL PARALLELISM IMPLEMENTATION

To profit from our multicore DSP architecture and the potential parallelism in H264/AVC encoder, Frame Level Parallelism approach is chosen to parallelize data processing and accelerate encoding run-time. Our choice is based on several reasons: First, this approach is characterized by high scalability. Second, this method ensures an important encoding speedup without inducing any rate distortion (Quality degradation or bit-rate increase) as the case of slice Level Parallelism method. Finally, it presents a compromise between GOP Level Parallelism and MB Level Parallelism implementations in terms of memory amount, CPU synchronizations cost, load balance, and implementation complexity.

A. H264/AVC Video encoding demo

To perform real-time video encoding demo, frames acquisition should be also performed in real-time. For that, 277Mbits/s as transmission bandwidth at least is required to transfer HD frames @ 25 f/s in YCrCb 4:2:0 format ((1280x720x1.5)x8bits x25f/s). As 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 preliminary step to send raw frames to DSP as shown in Fig. 4. In fact, Our DSP platform and the used PC include both a Gigabit Ethernet communication interface which makes it possible to perform real-time data transfer between them.

As our platform includes eight DSP cores, we reserved the first core "core0" to handle data transfer between the DSP and the camera board side (PC) via Ethernet connection. It is considered as a master processor that executes a TCP/IP server application (transmission Control Protocol/Internet Protocol) exploiting Network Developer's Kit library [8]. It is used firstly to receive the current frames sent by the camera board side and save them into the DDR3 memory which is a shared memory for all DSP cores. Consequently, the seven remaining cores are considered as slaves and they are devoted to encode the seven received frames in parallel way.

For each slave core (1 to 7), a memory section is allocated in the DDR3 memory. It contains the current frame, the reconstructed frame (RECT), and the bit-stream.

Our H264/AVC program is loaded into each internal L2 memory of the seven cores. Local variables used during encoding such as predicted MB buffers, transform and quantification matrixes, and best predicted modes etc are all allocated also in the L2 memory of each core to avoid data overlap among different cores. Our H264/AVC single core implementation is based on MBs row Level implementation [9]. It is an optimized design for the encoder in order to profit from the internal on-chip L2 memory and reducing DDR3 accesses. It consists in copying one MBs row instead of one MB (the classic encoding way) from the current frame in the DDR3 into a MB row buffer in the internal L2 memory.

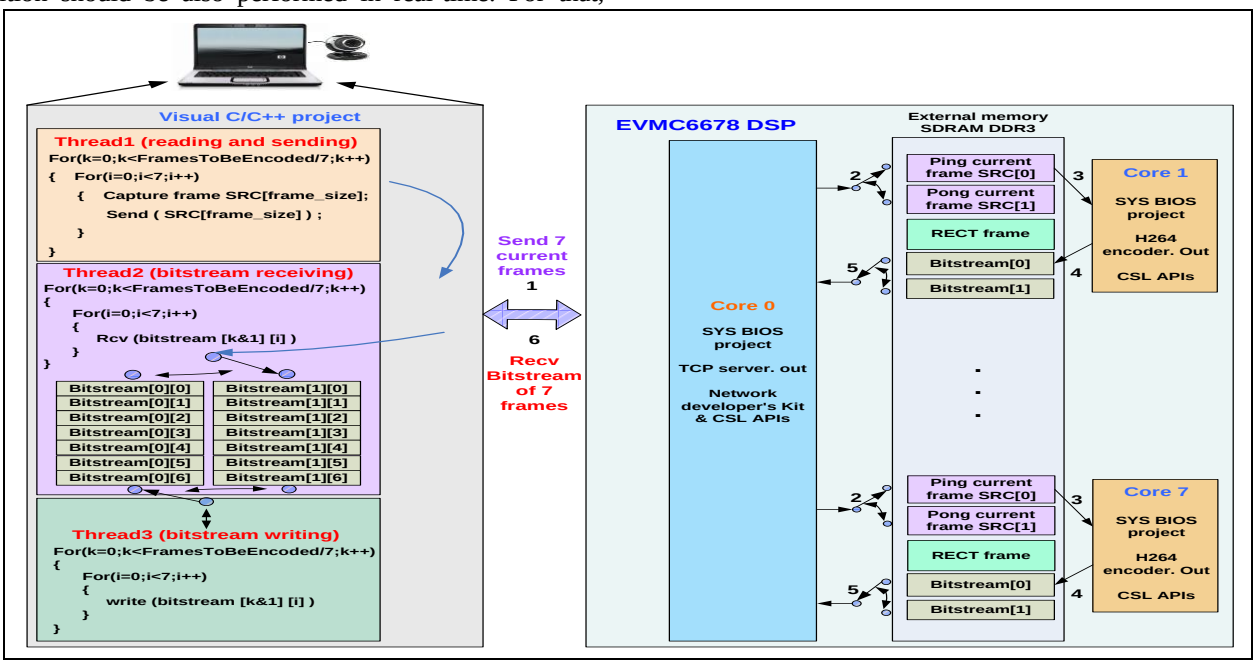

Fig. 4. H264/AVC vide encoder demo using Frame Level Parallelism approach

Encoding process will be thereafter performed between the CPU and the fast L2 memory without accesses to the DDR3. This implementation requires copying three MBs rows from the reconstructed frame into L2 memory to be used as motion estimation search window for all the MBs of the current MBs row. More details of this encoder design are presented in our previous publication [9].

Once encoding process is achieved, core0 sends the bit-streams of all encoded frames to the PC in order to store them in a file or decode them later.

To improve the classic Frame level parallelism implementation, our optimization consists in hiding communication overhead. The proposed enhancement is based on two strategies: the first is exploiting the ping-pong buffers technique on the DSP side in order to overlap encoding process with reading and writing data processes. The second is using a multi-threading approach on the camera board side as shown in Fig. 4. Thus, three threads are created to handle: 1) Reading raw frames and sending them to DSP via Ethernet. 2) Receiving bit-streams from DSP. 3) Saving the received bit-streams in a file.

On the DSP side, for each slave core, a ping-pong buffer is allocated for both the current frame and the generated bit-stream. A single buffer is used for the reconstructed frame since no transfers are required for this data.

OpenCv library [10] is exploited in our C/C++ project on the camera board side in order to capture raw frames from the HD USB camera. This C++ library is also used to convert captured frames from RGB to YCrCb 4:2:0 format that is required in our H264/AVC video encoder.

B. Parallel encoding steps using enhanced FLP approach on seven DSP cores

Encoding steps of our parallel implementation using the enhanced FLP approach on seven DSP cores are described in Fig. 5 and detailed as follows:

- Thread1 captures the first frame from the camera and sends it to DSP. Core0 receives this frame and stores it into the ping buffer SRC[0] of core1. After that, Core0 notifies core1 by sending an Inter Processor Communication event (IPC) to inform it that it can start encoding its current frame.
- When receiving the IPC interruption from core0, core1 starts the encoding process. At the same time, thread1 continues reading and sending the next six frames to core0 which will store them into the ping buffers of core2 to core7. Thus, each core immediately starts encoding its corresponding frame after receiving the IPC event from core0.
- Core1 is the first core that starts encoding process. Once it finishes encoding the first 3 MBs rows of its current frame, it sends an IPC to the following core (core2) which itself is in a wait state for an interruption from core1 to start encoding its appropriate frame in order to respect temporal dependency by using these 3 MBs rows as a search window for motion estimation. The same procedure will be reproduced from core3 to core7.
- To avoid that core *i* exceeds core *i-1* (which is strongly possible because encoding load balance is not equal among successive frames so, it may lead to an erroneous result), encoding the next MBs row by core *i* is conditioned with the reception of an IPC from its previous core. Thus, each core sends an IPC to its following core after encoding a MBs row which its index is higher than 3.
- Since each core starts encoding its MBs row when its previous core finishes encoding the first three MBs rows, it should not wait an IPC from its antecedent to encode the two last MBs rows of the current SRC frame. Otherwise, encoding will be blocked by waiting an incoming IPC. In fact, each core sends Max_MBs_rows-2 IPC interruptions to its following core.
- During encoding the first seven frames by core1 to core7, thread1 sends the next 7 frames to core0 which will store each frame into the pong buffer SRC[1] of each core. Because encoding process takes more time than reading process, communication delays are hidden and they do not contribute to the parallel run-time.
- When core *i* terminates encoding its current frame, it stores the bitstream into the ping buffer bitstream[0]. Then, it notifies core0 by sending an IPC event to inform it that it can forward its bitstream to the PC. Immediately after that, core *i* starts encoding its pong frame already stored into SRC[1] without any wait and stores the bitstream into the pong buffer bitstream[1] in order to not overwrite data stored into bitstream[0] which is being transferred.
- While core *i* encodes its pong frame, core0 sends the ping bitstream[0] corresponding to core *i* without waiting that all cores finish encoding their respective frames. Thread2 receives the ping bitsteams and stores them into the ping buffers Bitstream[0][i]. At this time, thread3 writes the bitstreams in a file and thread1 sends the next seven frames to core0 which will store each frame into the ping buffer SRC[0] of each core.
- With this enhancement, the pong SRC frames encoding, the ping bitstreams writing, and the next 7 ping SRC frames reading and sending are processed in parallel way.
- The processing is thereafter looped in a reverse order for SRC frames and bitstreams through ping pong buffers.
- As shown in Fig. 5, no significant delays are occurred. Each core processes its respective data without any waiting time. Multithreading algorithm with ping pong buffers technique efficiently overlap data transfer with encoding process.

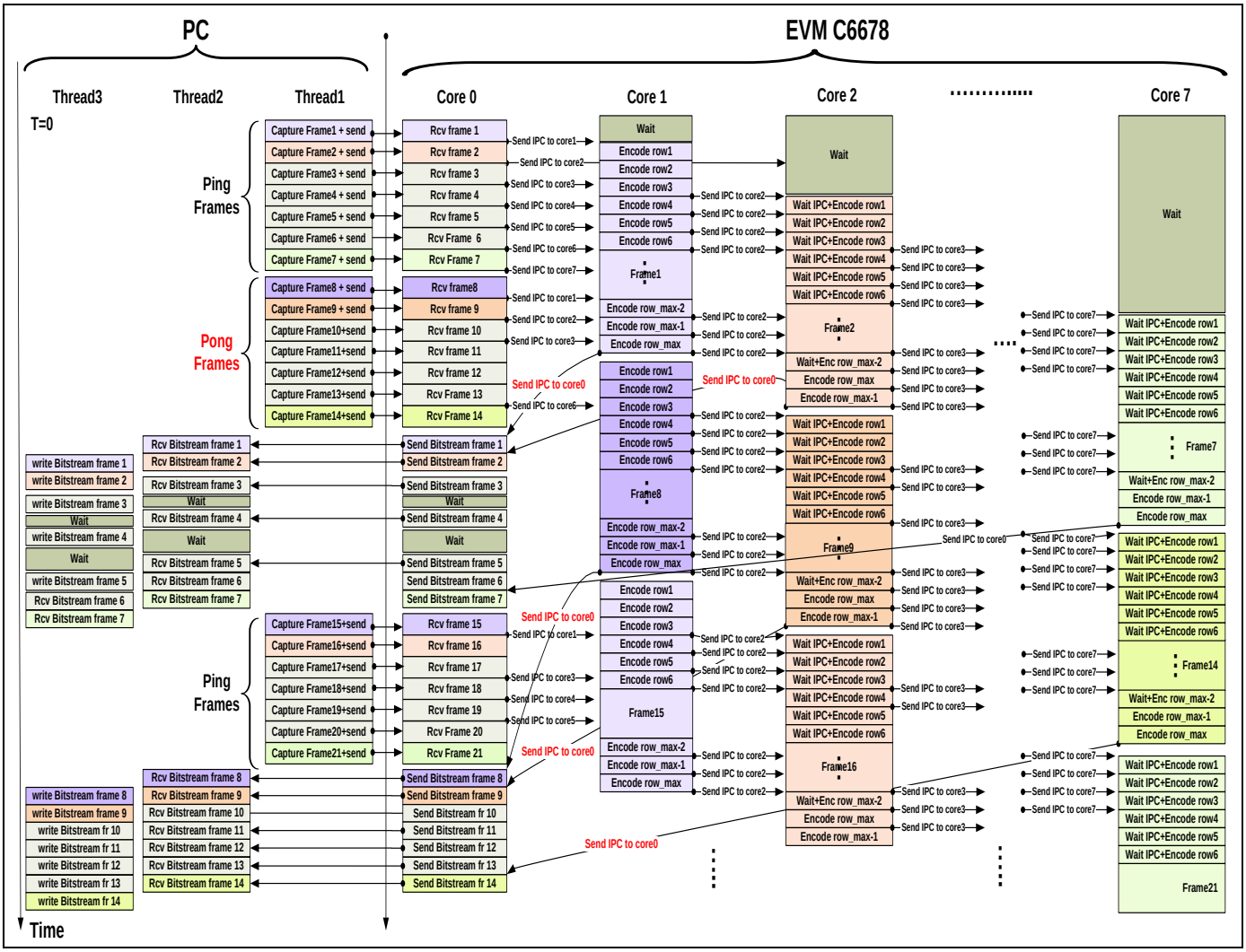

Fig. 5. chronological steps of the Enhanced FLP implementation on the multicore DSP TMS320C6678

C. *Cache coherence*

Parallel processing on a multicore platform may lead to a cache coherence problem. This is happen when processing a shared data by several cores with a private cache memory. In order to deal with cache coherence problem, the Chip Support Library included in the MCSDK development Kit [11] of TI provides two API functions:

- CACHE_wbL2() to return back the cached data from the cache memory to its original location in the shared memory.
- CACHE_invL2() to invalidate the cache lines and force the CPU to read data from its original location in the shared memory.

In our case, when receiving the captured frames from the PC, core0 should write back the cached lines to their locations in the external DDR3 memory in order to be encoded later by the slave cores. Similarly, core1 to core7 should invalidate the current frames addresses in their cache memories before start encoding. This allows processing the updated data written by core0 and not the old data already existed in their cache memories. Furthermore, when encoding process is achieved, core1 to core7 should write-back the bit-streams from their cache memories to their original locations in the external memory. Consequently, core0 should invalidate the bit-streams from its cache memory in order to send the updated values to the PC.

D. *Experimental results*

Our parallel implementation of H264/AVC encoder is performed on the eight-core TMS320C6678 DSP. Each core of our target is running @ 1 GHz of CPU frequency. H264/AVC LETI laboratory's software which is an optimized version of the Joint Model (JM) software [8] is used. Experimental simulations are performed on the most commonly used video test sequences in HD (1280x720) resolution. These sequences are a raw data in YCrCb 4:2:0 format recommended by the Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG organizations [12]. Encoding parameters are presented in TABLE I.

TABLE I. ENCODING PARAMETERS

Intra period (GOP size)	8
Search range	16
QP value	30
Frames to be encoded	280
Error metric for mode decision	Sum of Absolute Difference
Entropy coding	CAVLC
Rate control	off
Number of Reference Frames	1

For performance evaluation, encoding speed is measured for several implementations using different number of cores and expressed as follows:

$$\text{encoding speed (f/s)} = \frac{\text{DSP frequency}}{\text{Number of clock cycles}} \text{ x number of frames} \quad (1)$$

In our experiments, data transfer time, which includes PC frames capturing, sending them to DSP via Ethernet, and loading them to DSP memory by core0, is considered in our computation in order to evaluate our enhancement techniques of hiding communication overhead. Table II shows the encoding speeds of our optimized H264/AVC video encoder implementation with different number of cores.

TABLE II. ENCODING SPEEDS FOR HD RESOLUTION

Video sequences	**Enc speed on 1 core (f/s)**	**Enc speed on 3 cores (f/s))**	**Enc speed on 5 cores (f/s)**	**Enc speed on 7 cores (f/s)**
Planet	4.19	12.07	18.92	26.83
Parkjoy	4.78	12.98	21.81	28.87
Nature	3.88	11.01	17.58	25.75
Mob_cal	4.01	11.24	18.28	26.18
Crowdrun	3.79	10.80	17.13	25.12
Birds	4.98	12.37	22.04	30.46
Shields	3.87	11.10	17.68	25.15
Stockholm	3.98	11.20	17.32	25.34
Average	**4.18**	**11.60**	**18.84**	**26.71**

Experimental results show that single core implementation is not able to meet the real-time encoding compliant (25 f/s). In fact, the obtained encoding speed using a single DSP core is equal to 4,18 f/s in average for HD resolution. Applying our multicore implementation exploiting the enhanced FLP approach allows enhancing the encoding speed. As shown in Table II, encoding speed is significantly increased from 4,18 f/s to 11,6 f/s and 18,84 f/s when using respectively 3 and 5 DSP cores for encoding. Exploiting seven DSP cores allows achieving the real-time processing and reaching 26.71 f/s in average surpassing the real-time constraint 25 f/s. Our parallel implementation accelerates the encoding run-time by a factor of 6,38 without inducing any rate distortion in terms of visual quality degradation or bit-rate increase compared to single core implementation.

Despite that data transfer time has been considered in our performance evaluation; this time does not contribute in the encoding run-time. This affirms that our proposed data transfer scheduling technique efficiently hides communication overhead. In fact, data transfer time will be only noticed in the first seven frames as shown in Fig. 5 but after that, these transfers are mostly overlapped with the encoding process especially that this latter takes more time than reading and writing processes.

V. CONCLUSION

In this paper, an embedded real-time H264/AVC video encoder implementation on the last generation of TI's multicore DSP TMS320C6678 was presented. FLP approach was selected to accelerate the encoding run-time. Exploiting a multi-threading algorithm combined with using a ping-pong buffers technique allows enhancing our implementation and efficiently hides communication overhead. Experimental results on seven DSP cores running each @ 1 GHz showed that real-time was achieved by reaching up to 26 f/s as encoding speed for HD resolution. Our parallel implementation accelerated the encoding process by a factor of 6,38 without inducing any visual quality degradation or bit-rate increase. As perspective, we will exploit our expertise in multicore DSP implementation to implement the newest HEVC video encoder and trying to achieve real-time encoding.

REFERENCES

[1] Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG, "Advanced video coding for generic audiovisual services," Avril 2013.

[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] 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, September2013..

[4] 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.

[5] António Rodrigues, Nuno Roma, and Leonel Sousa," p264: Open Platform for Designing Parallel H.264/AVC Video 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, 2010.

[6] 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 12th IEEE International Conference , pp.465,470, 1-3 Sept. 2010

[7] TMS320C6678 manual datasheet, online available: http://www.ti.com/lit/ds/symlink/tms320c6678.pdf

[8] TI NDK user's Guide, online available: http://downloads.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/ndk/2_21_01_38/exports/ndk_2_21_01_38/docs/spru523h.pdf

[9] Bahri, N., Werda, I., Grandpierre, T., Ben Ayed, M., Masmoudi, N., & Akil, M, "Optimizations for Real-Time Implementation of H264/AVC Video Encoder on DSP Processor," International Review on Computers & Software,8(9), 2013.

[10] OpenCv library: http://opencv.org/

[11] MCSDK user's guide, online available: http://processors.wiki.ti.com/index.php/BIOS_MCSDK_2.0_User_Guide

[12] 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