HAL
open science

# Real-time H264/AVC encoder based on enhanced frame level parallelism for smart multicore DSP camera

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

## To cite this version:

Nejmeddine Bahri, Nidhameddine Belhadj, Thierry Grandpierre, Mohamed Ali Ben Ayed, Nouri Masmoudi, et al.. Real-time H264/AVC encoder based on enhanced frame level parallelism for smart multicore DSP camera. Journal of Real-Time Image Processing, 2014, 12, pp.791-812. 10.1007/s11554-014-0470-6 . hal-01192770

**HAL Id: hal-01192770**

**https://hal.science/hal-01192770**

Submitted on 7 Sep 2015

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Real-time H264/AVC Encoder based on Enhanced Frame Level Parallelism for Smart Multicore DSP Camera

Nejmeddine Bahri(1), Nidhameddine Belhadj(1), Thierry Grandpierre(2), Mohamed Ali Ben Ayed(1), Nouri Masmoudi(1), Mohamed Akil(2)

(1)National school of Engineers, LETI Laboratory, University of Sfax, Tunisia

(2)ESIEE Engineering, LIGM Laboratory, University Paris-EST, France

nejmeddine.bahri@gmail.com, Thierry.grandpierre@esiee.fr, nouri.masmoudi@enis.rnu.tn, mohamed.akil@esiee.fr

**Abstract** The latest generation of multicore Digital Signal Processors (DSP), their high computing power, low consumption and integrated peripherals will allow them to be embedded in the next generation of smart camera. Such DSPs allow designers to evolve the vision landscape and simplify the developer's tasks to run more complex image and video processing applications without the need to burden a separate Personal Computer (PC). This paper explains how exploiting the computing power of a multicore DSP TMS320C6472 in order to implement a real-time H264/AVC video encoder. This work prepares the way to the implementation of the new High Efficiency Video Coding standard (HEVC-H265). To improve encoding speed, the enhanced Frame Level Parallelism (FLP) approach is presented and implemented. A real-time fully functional video demo is given taken into account video capture and bitstream storage. Experimental results show how we efficiently exploit the potentials and the features of the multicore platform without inducing PSNR degradation or bitrate increase. The enhanced FLP using five DSP cores achieves a speedup factor of 4.3 times in average compared to a mono-core processor implementation for Common Intermediate Format (CIF 352x288), Standard Definition (SD 720x480) and High Definition (HD 1280x720) resolutions. This optimized implementation allows us to exceed the real-time by reaching an encoding speed of 98 f/s (frame/second) and 32 f/s for CIF and SD resolutions respectively and saves up to 77% of encoding time for the HD resolution.

**Keywords** H264/AVC encoder, DSP, multi-core, Frame Level Parallelism, real-time.

## 1 Introduction

Nowadays, smart cameras or machine vision solutions [1], [2] need to run complex image and video processing applications on growing amounts of data while meeting hard real-time constraints. New technologies of programmable processors such multicore DSPs, embedded heterogeneous systems (ARM-DSP [3], DSP-FPGA, ARM-FPGA), offer a very promising solution for these applications that require high computing performances. They are characterized by a high processing frequency with low power consumption compared to General Purpose Processor (GPP) or Graphic Processor Unit (GPU). Several manufactures [4] such as Freescale [5] and Texas Instruments (TI) [6] solve the challenges of smart cameras with their high performance multicore DSP processors. Exploiting these embedded technologies, smart cameras are changing the vision landscape and pushing developers to run several applications without the need to use any connected PC. In the area of video application, compression represents an interesting task among the main applications of smart camera or machine vision in addition to other tasks such object detection, tracking, recognition…etc. The commercialized encoding IPs allow real-time performance but lack in flexibility. In fact, they cannot be upgraded to follow the latest protocol enhancements and the latest advances in video compression. Actually, a new video coding standard is appeared on the market which is the HEVC-H265 but in the side, there are several smart cameras still work until now with old video coding standard as motion JPEG or MPEG4. So it is time now to follow developments in this field.

DSPs offer software flexibility that is important to allow upgradability. They allow us to build highly flexible and scalable cameras that can follow the latest advances in video compression. Encoder parameters can also be finely tuned depending on the application's requirements. They are also characterized by relatively low software development cost and time-to-market reduction compared to ASIC development or FPGA implementation that requires a tremendous VHDL expertise which may not deal with time-to-market constraint.

In this context, the TI's high performance multicore DSP processor TMS320C6472 is used in order to achieve a real-time implementation for the H264/AVC [7] video encoder. This work will be our start point for the new video standard HEVC [8]. Effectively; since HEVC encoder keeps the majority of H264/AVC features (GOPs, frames, and slices structures) our proposed approach will also benefit for our future H265 implementation.

H264 encoder is characterized by high coding efficiency comparing with previous standards. However, this efficiency is accompanied by a high computational complexity that requires a high-performance processing capability to satisfy real-time constraint (25 to 30 f/s).

When moving to high resolutions, encoding time is drastically increased. Frequency limitation of embedded mono-core processor makes it hard to achieve real-time encoding especially for HD resolutions. Using parallel and multicore architectures will be crucial to reduce the processing time of H264/AVC encoder.

Several works have been published exploiting the potential parallelism of H264/AVC standard by applying a functional partitioning algorithms, data partitioning algorithms or both. Multi-processor, multi-core, multi-threading encoding system and parallel algorithms have been discussed in many papers [9] to [27]. This paper presents the Frame Level Parallelism (FLP) approach and describes its complete implementation in a H.264/AVC encoder using a multicore DSP TMS320C6472.

The remainder of this paper is organized as follows: next section provides an overview of data dependencies and parallelism in H.264/AVC standard. Section 3 details the related works on the parallel implementations of H264/AVC encoder. The internal architecture of our multicore DSP TMS320C6472 is described in Sect.4. Section 5 presents our optimized implementation of H264 encoder on a single DSP core. Section 6 focuses on the FLP algorithm implementation on five DSP cores. It details the whole coding chain (image capture, bitstream transfers), and finally gives experimental results. The best approach, based on the enhanced FLP is detailed in Sect.7 which also includes experimental results. Finally, section 8 concludes this paper and presents some perspectives.

## 2 Overview of data dependencies and parallelism in H264/AVC encoder

The H.264/AVC encoder baseline profile is a video compression standard used to reduce the video data amount in order to overcome the limitation of transmission bandwidth and the huge amount of memory requirement for storing high definition video sequences. This standard consists of several functions in order to generate the compressed bitstream of the input video as shown in Fig.1.

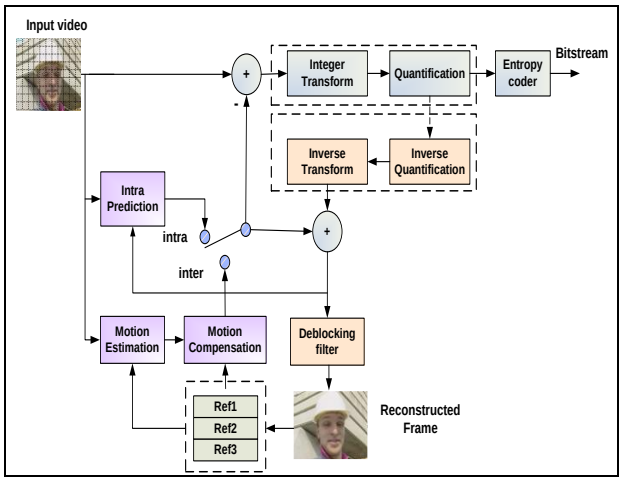


**Fig. 1** H264/AVC video encoder strucure

This standard divides a video sequence into a hierarchical structure with six levels as shown in Fig. 2. The top level of this structure is the sequence that contains one or more groups of pictures (GOP). Each GOP is composed of one or more frames. Finally, the frames are divided into one or more independent slices, subdivided themselves into macroblocks of 16x16 pixels (MB) and blocks of 4x4 pixels. Each MB undergoes two prediction types: 1) intra prediction: it consists of performing intra16x16 and intra4x4 prediction modes in order to reduce spatial redundancies in the current frame. 2) inter prediction: it consists of determining the motion vector of the current MB relative to its position in the reference frames. It includes 7 prediction modes in order to reduce temporal redundancies existed among successive frames. A mode decision is then performed to select the best prediction mode. Integer transform and quantification modules are performed on the best predicted MB in order to keep only the most significant coefficients. An entropy coding is finally performed to generate the compressed bitstream. A decoding chain is included in the encoder structure in order to keep the reconstructed frame that will be filtered with a de-blocking filter in order to eliminate artifacts. The reconstructed frame will be used as a reference for the next frames to perform motion estimation.

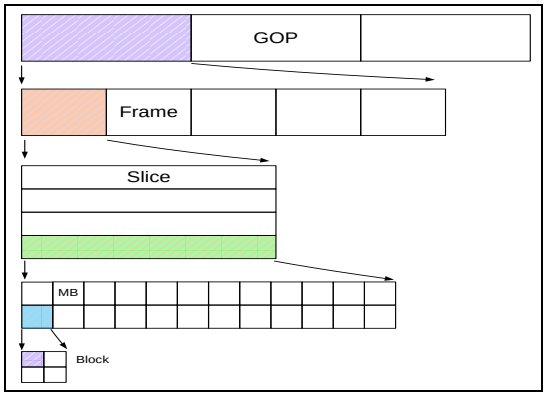


**Fig. 2** Hierarchical decomposition of an H.264 video sequence

According to functions organization and hierarchical sequence structure in H.264/AVC encoder, there are mainly two partitioning families:

**Task-level parallelization (TLP) or functional partitioning:** it consists of splitting the encoder into several steps, identify them into a different group of tasks equal to the number of threads available on the system and run these groups of tasks simultaneously as a pipeline. Thus, the appropriate functions that could be grouped together to be processed in parallel and the other functions that will be executed in serial to respect data dependencies should be efficiently chosen. Also, tasks computational complexities should be taken into consideration in order to maximize the encoding gain and ensure a workload balance between the parallel tasks. Finally, when grouping functions, synchronization overhead should be minimized as much as possible by eliminating data dependency between the different function blocks. For example intra prediction modes (13

modes) and inter prediction modes (7 modes) could be processed in parallel because no dependencies existed among them. In the other side, integer transform, quantification and entropy coding have to be processed in serial way given the dependencies among them.

**Data-level parallelization (DLP) or data partitioning**: it exploits the hierarchical data structure of H264/AVC encoder by simultaneously processing several data levels on multiple processing units. DLP is limited by data dependencies among different data units.

For H264/AVC encoder, there are two major types of data dependencies:

**Spatial dependencies:** they exist amongst macroblocks within the current encoding frame. In fact, to perform intra prediction modes, motion vector prediction and reconstructed MB filtering for the current MB, such data are required from its neighboring MBs (Left, Top Left TOP and Top right) already encoded as shown in Fig. 3. So, the current MB could be encoded only if its neighboring MBs have been encoded.

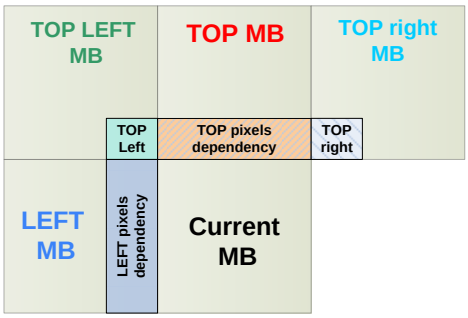


**Fig. 3** spatial dependencies for the current MB

**Temporal dependency:** to determine the motion vector of the current MB in relative to its position in the previous encoded frames, a motion estimation (ME) algorithm such as MB matching is performed. The search of the corresponding MB is restricted in a specific area called the "search window" in the reference frames (the previous encoded frames) instead of scanning the whole frame in order to reduce the computing complexity. So a partial dependency among MBs of successive frames is imposed and limited to the search windows.

As data partitioning is restricted by these data dependencies, several points could be noticed. No dependencies existed among different GOPs because each GOP is started by an intra frame “I” where only intra prediction is performed, so dependencies is existed only among MBs in the same frame. The remaining frame of the GOP are a predicted frames “P” where both intra and inter prediction are performed. Hence, several GOPs could be encoded in parallel. This method is called GOP Level Parallelism [14]. A partial dependency is existed between successive frames of the same GOP due to motion estimation in the search window. Thus, multiple frames could also be encoded in pipeline once the search window is encoded and this method is called Frame Level Parallelism [12]. When dividing frame into independent slices, several slices could be processed in parallel and this approach is called slice level parallelism [13]. Finally, in the same frame, multiple MBs could be encoded at the same time once its neighboring MBs are already encoded. This scheme is called MB level Parallelism [12].

## 3 Related works

To overcome the high complexity of H264/AVC encoder and to resolve the problem of mono-core processor frequency limitation, many researchers have been worked on the parallelism of H264/AVC encoder in order to meet the real-time constraint and achieve a good encoding speedup which can be presented by the following equation.

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

Several implementations exploiting multi-threads, multi-processors and multicore architectures are discussed in many papers:

Zhibin Xiao et al. [9] exploit task level parallelism approach. They partition and map the dataflow of H.264/AVC encoder to an array of 167-core asynchronous array of simple processors (AsAP) computation platform coupled with two shared memories and a hardware accelerator for motion estimation. They process the luminance and the chrominance components in parallel. Intra4x4 modes and intra16x16 modes are calculated 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 transform and quantification and 17 processors for CAVLC. A hardware accelerator is used for motion estimation. Despite all these hardware resources, a real-time implementation is not achieved. The presented encoder is capable of encoding VGA (640 x 480) video at 21 frames per second (fps). Reducing the number of candidate modes for intra4x4 and intra16x16 induces visual quality degradation and bitrate increase.

Sun et al. [10] implement a parallel algorithm for H.264 encoder based on MB region partition (MBRP). They split the frame into several MB regions composed by adjoining columns of MBs. Then, they map the MB regions onto different processors to be encoded satisfying data dependencies in the same MBs row. Simulation results on 4 processors running at 1.7 GHz show that the proposed partitioning achieves a speedup by a factor of 3.33 without any rate distortion (Quality, Bitrate) compared to H264 software JM10.2 [11]. In the other side, they are still far from real-time implementation that requires at least 25 f/s. They can encode only 1frame/1.67s for CIF resolution and 1frame/6.73s for SD resolution.

Zhuo Zhao et al. [12] propose a new wave-front parallelization method for H.264 encoder. They mix two partitioning methods: MB row level parallelism and frame level parallelism. All MBs in the same MB row are processed by the same processor or thread to reduce data exchanges between processors. MBs in different frames can be processed concurrently if the reconstructed MBs

in the reference frame forming the search window are all available. They implement this method using JM9.0 on a Pentium 4 processor running at 2.8 GHz. Simulations on 4 processors prove that a speedup by a factor of 3 is achieved (3.17 for QCIF resolution (Quarter CIF 176x144) and 3.08 for CIF). Encoding quality was not changed and it remains the same as the original software JM 9.0. In the other side, the runtime is far from real-time implementation. In fact, only 1frame/1.72s is encoded for CIF resolution.

Yen-Kuang et al. [13] parallelize the H.264 encoder exploiting thread-level parallelism using OpenMP programming model. Slice level partitioning is performed on 4 Intel Xeon™ processors with Hyper-Threading Technology. Results show a speedups ranging from 3.74x to 4.53x. The drawback of slice parallelism is that it affects the rate distortion performance. Indeed, it provides PSNR degradation and an important increase in bitrate especially when the frame is decomposed into several independent slices.

S.Sankaraiah et al. [14] [15] apply the GOP level parallelism using multithreading algorithm in order to avoid data dependencies. 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. The drawback of GOP level parallelism is its very high encoding latency that is not compatible with video conference applications.

Rodriguez et al. [16] go a step further and propose an implementation of H.264 encoder using GOP level parallelism combined with slice level parallelism on a clustered workstations using Message Passing Interface (MPI). The first approach speeds up the processing time but provides a high latency and the second approach is used to reduce this latency by dividing each frame into several slices and distributing these slices to computers belonging to a subgroup of computers. With this technique, the encoding latency is relatively reduced. However, increasing the number of slices per frame has significant adverse effects on the rate distortion (bitrate increment). Also, clustered workstations are a costly solution and they are not intended for embedded applications.

Shenggang Chen et al. [17] introduce an implementation of an on-chip parallel H.264/AVC encoder on hierarchical 64-cores DSP platform. This platform consists of 16 super nodes (4 DSP cores for each node). 2D WaveFront algorithm for macroblock level parallelism is used and one macroblock is assigned to one super node. Subtasks for encoding one macroblock 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 wavefront parallel algorithm does not introduce any quality loss; however, the used CABAC-based bitrate estimation and parallel CABAC evolutional entropy coder cause a bitrate increment. Real-time processing is not given in this paper.

Ming-Jiang Yang et al. [18] implement the H264/AVC encoder on the dual-core DSP processor ADSP-BF561 chipset using functional partitioning. Core A of the BF561 processor is dedicated to perform mode decision, intra prediction, motion compensation, integer transform (IT), quantization, de-quantization, inverse integer transform, and entropy encoding. Core B is assigned to perform in-loop filtering, boundary extension, and half-pel interpolation. Core A and core B execute tasks in two pipeline stages. The proposed encoder system achieves real-time encoding for CIF resolution but not for higher resolutions (VGA, SD and HD).

Zrida et al. [19] present a parallelization approach for embedded Systems on Chip (SoCs). It is based on exploration of task and data levels parallelism, the parallel Kahn process network (KPN) model of computation and the YAPI programming C++ runtime library. The used SOC platform relies on 4 MIPS processors. Simulation results of this work show that a speedup of 3.6 is achieved for QCIF format but real-time encoding is not reached even for low resolutions (7.7 f/s for QCIF format).

António Rodrigues et al. [20] implement 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 2.7GHz. Two parallelism levels are combined: slice level and macroblock level. A multi-threading algorithm using openMP is used for JM software. The frame is decomposed into slices; each slice is processed by a group of cores. Several MBs in the same slice are encoded in parallel with respecting of data dependencies 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 cores used for encoding. Using a MB level Parallelism requires that data have to be shared which leads to a memory bottleneck and higher latency. Also, increasing the number of slices introduces a bitrate distortion. Real-time is not noticed in this work.

Olli lehtoranta et al. [21] implement a row-wise data parallel video coding method on the quad TMS320C6201 DSP system. The frame is decomposed into slices by row-wise and each slice is mapped to a DSP slave. A DSP master is devoted to swap data to/from DSP slaves. The real-time 30f/s is reached only for CIF resolution but not yet for higher resolutions. The main drawback of this approach is an increase in bitrate and PSNR degradation because of using slice level parallelism.

In [22], author develops a parallel implementation of the intra-prediction H.264/AVC module by using the computational resources of a GPU and exploiting the Compute Unified Device Architecture (CUDA) programming model. They apply two partitioning

methods: 1) data partitioning by processing the luminance Y and the two chroma components Cr, Cb in different parallel tasks. 2) Task partitioning by processing the intra prediction 16x16, intra prediction 4x4 and chroma intra prediction in parallel. This implementation of the intra prediction module achieved a speedup of 11x, relatively to the sequential implementation of the reference software but as we know the inter prediction is the most important module in the H264/AVC encoder which takes the lion's share of processing time. So, preferably, this module should be accelerated in addition to intra prediction. Moreover, processing chrominance data in parallel with the luminance component does not give a significant speedup if we know that chrominance processing is negligible relatively to luminance processing. Finally, the luminance and chrominance processes are not totally independent; thus; a dependency is existed among the two components during filtering and entropy coding processes which leads to a high latency.

Fang Ji et al. [23] propose a H264/AVC encoder on an MPSOC platform using GOP level parallelism approach. They build three Microblaze soft cores based on XILINX FPGA. A main processor is devoted to prepare the frames into the shared memory. Then, each processor among the remaining coprocessors will encode its appropriate GOP. Experiments show that the average speedup is 1.831. The problem of the GOP approach is its higher latency. Real-time is not achieved. This solution encodes only 3 f/s for QCIF resolution.

Xiulian Peng et al. [24] propose a pure line-by-line coding scheme (LBLC) for intra frame coding. The input image is processed line by line sequentially, and each line is divided into small fixed-length segments. The encoding of all segments from prediction to entropy coding is completely independent and concurrent at many cores. Results on a general-purpose computer illustrate that the proposed scheme can get a 13.9 as speedup factor with 15 cores but in the other side, this method affects the rate distortion because of discarding the Left dependency for each MB.

Huayou Su et al [25] propose a parallel framework for H.264/AVC based on massively parallel architecture implemented on NVIDEA's GPU using CUDA. They present several optimizations to accelerate the encoding speed on GPU. A parallel implementation of the inter prediction is proposed based on a novel algorithm MRMW (Multi-resolutions Multi-windows) that consists of using the motion trend of a lower resolution frame to estimate that of the original frame ( higher resolution). The steps of MRMW are parallelized with CUDA on the different cores of the GPU. Also they perform a multilevel parallelism for intra-coding. For that a multi-slice method is introduced. Each frame is partitioned into independent slice. At the same time, the wave-front method is adopted for parallelizing the MBs in the same slice. Some dependencies within MBs are not respected to maximize the parallelism. Moreover, CAVLC coding and filtering processes are also parallelized by decomposing these modules into several tasks. Experimental results show that about 20 times the speedup can be obtained for the proposed parallel method when compared to the reference program. The presented parallel H.264 encoder can satisfy the requirement of real-time HD encoding of 30 fps. In the other side, this implementation affects the visual quality by inducing a PSNR degradation ranging from 0.14 dB to 0.77 dB and a little increase in bitrate because of using multi-slice parallelism and some dependencies are not respected.

O. Adeyemi et al [26] presents a 4kUHD video streaming over wireless 802.11n. They perform the entire encoding chain including 4K camera capture, YUV color space conversion, H264 encoding using CUDA on NVIDIA Quadro 510 GPU and real-time live streaming. To speed up the encoding, several modules are parallelized such intra and inter prediction modules by exploiting a dynamic parallel motion algorithm and a novel intra prediction mode. Also, they used a Zero-Copy memory allocation technique to reduce memory copy latencies between the host memory (CPU) and the GPU memory. Experiments confirm that 4kUHD real-time encoding for live streaming at low bitrates is possible. A little deviation on visual quality is induced. Despite this, GPUs are a costly solution and they are not suitable for embedded applications because of high power consumption compared to other platforms.

Wajdi Elhamzi et al [27] present a configurable H264 motion estimator dedicated to video codec on a smart camera accelerator based on Virtex6 FPGA component. They propose a flexible solution to adjust the video stream transferred by the smart camera. The accelerator is able to support several search strategies at IME (Integer Motion Estimation) stage and different configurations for FME (fractional Motion Estimation) stage. Experiments show that the obtained FPGA based architecture can process IME on 720x576 video streams at 67 fps using full search strategy. FPGA solution remains an interesting way to achieve real-time processing but when moving to implement the whole H264 encoder, a huge FPGA surface and a lot of design and compilation time with tremendous VHDL expertise are required which may not deal with time-to-market constraint. Finally, the low hardwired block frequency and bus bandwidth for data transfers between processor and accelerator represent the major drawbacks of FPGA implementations.

## 4 DSP platform description

Software flexibility, low power consumption, time-to-market reduction, and low cost make DSPs an attractive solution for embedded systems implementations and high performance applications. Motivated by these merits and encouraged by the great evolution of DSP architectures, we chose to implement the H264/AVC encoder on a low cost multicore DSP TMS320C6472 to profit from high processing frequency and an optimized architecture in order to achieve real-time embedded video encoder.

TMS320C6472 DSP [28] belongs to the latest generation of multicore DSPs made by Texas Instrument. Low power consumption and a competitive price tag make the TMS320C6472 DSP ideal for high-performance applications and suitable for many embedded implementations. Several benchmarks are performed by Texas Instruments to compare between DSP, General Purpose Processors (GPP) and Graphic Processor Unit (GPU) [29] [30]. These benchmarks demonstrate that the C6472 consumes 0.15 mW/MIPS (Million instructions per second) at 3 GHz. Also, at 3.7 watts per device, it offers even greater power savings compared to GPP in the same performance range. When the performance is distributed over power, the DSP is 4x more better than GPU and 18x than GPP.

As presented in Fig. 4, six C64x + DSP cores, very long instruction word (VLIW) architecture, 4.8 M-Byte (MB) of memory on chip, Single Instruction Multiple Data (SIMD) instruction set and a frequency of 700 MHz for each core are combined to deliver 33600 MIPS performance.

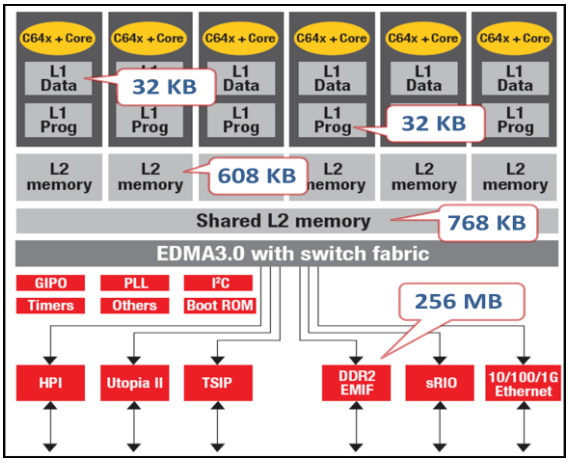


**Fig. 4** Internal architecture of TMS320C6472 DSP

Each C64x+ core integrates a large amount of on-chip memory organized as a two-level memory system. The level-1 (L1) program and data memories on this C64x+ core are 32 K-Byte (KB) each. This memory can be configured as mapped RAM, cache, or any combination of the two. The level 2 (L2) memory is shared between program and data space and is 608 KB in size. L2 memory can also be configured as mapped RAM, cache, or any combination of the two. In addition to L1 and L2 memory dedicated to each core, the six cores also share 768 KB of L2 shared memory. Shared L2 memory is managed by a separate controller and can be configured as either program or data memory. This large amount of on-chip memory may avoid access the external DDR2 memory, therefore reducing the power dissipation and accelerating algorithms processing since internal memory is faster than external memory. Performance is also enhanced using the EDMA (Enhanced Direct Memory Access) controller which is able to manage memory transfers independently from the CPU. Therefore, no additional overhead is caused when large data blocks are moved between internal and external memory. TMS320C6472 DSP supports different communication peripherals as Gigabit Ethernet for Internet Protocol (IP) networks, UTOPIA 2 for telecommunications and Serial RapidIO for DSP-to-DSP communications. This DSP consequently includes all the necessary components (DMA, RAM (Random Access Memory), input output management) required to communicate with a camera sensor. Note also that VLIW architectures are deterministic and dedicated to embedded hard real-time applications. This must be taken into account when compared to superscalar GPP (General Purpose Processors) based on expensive and consuming memory management units, out-of-order units etc.

Finally, it is clear that the power and the features of such DSP family perfectly fit the need of intelligent vision system embedded in smart cameras. It should also allow designers to build highly scalable camera that can follow the latest advances in video compression.

## 5 Optimized implementation on a single DSP core

Our choice for using this multicore DSP platform enables us to develop an academic H264/AVC codec [31] in our LETI laboratory (Laboratory of Electronics and Information Technologies) for future research and development targeting embedded video applications. Standard compliant LETI's codec was developed and tested first on a PC environment for validation and then migrated to TMS320C6472 DSP platform. This work will also benefit for our future H265 implementation.

To efficiently take advantages of multicore technology and the potential parallelism presented in the H264 standard, we must as a first step, elaborate an optimized H264/AVC architecture on a single DSP core and then move to a multicore implementation. This step consists of designing a data model that exploits DSP core architecture and especially internal memory which is faster than external SDRAM memory. Each core of TMS320C6472 DSP has 608 KB internal memory LL2RAM shared between program and data. Preferably and to the extent of possible, we should load both program and data within LL2RAM. For that reason, two implementations are proposed [32].

### 5.1 «MB level » implementation

This implementation is the conventional data structure processing in H264/AVC standard. It is based on encoding a MB followed by another MB until finishing the entire frame MBs. The principle of this first proposed architecture is detailed as follows: the program is loaded into internal memory LL2RAM. The current, the reconstructed, the reference frames and the bitstream are stored into external SDRAM memory regarding their important sizes for HD resolution. In order to avoid working directly with slow external memory, some data are moved into internal memory such as current MB, search window and reconstructed MB for the 3 $YC_rC_b$

components. The design of the MB level implementation is presented in Fig. 5. It highlights the memory allocations for the luminance components.

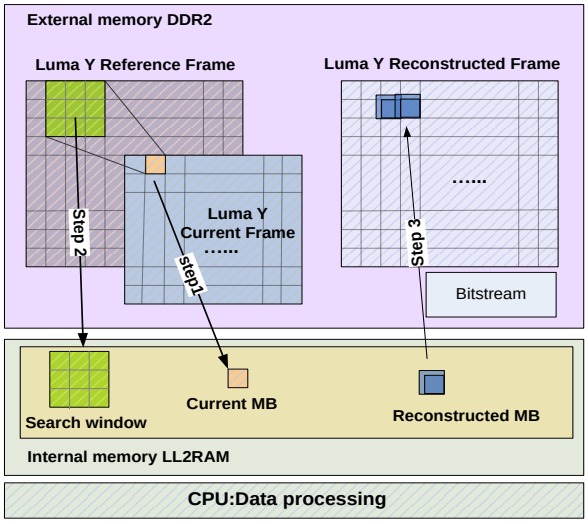


**Fig. 5** « MB level » implementation

The DSP core transfers the current MB (16x16) and the search window (48x48) respectively from the current and the reference frames from external to internal memory. Consequently, the data processing can be performed by the DSP core without external memory accesses. The reconstructed MB (20x20), extended by 4 pixels at the left and the top needed in the MB filtering, is transferred from the local memory into external memory at the reconstructed frame buffer. This process is repeated until completion of the entire current frame MBs. The most important advantage of this architecture is its adaptability to any DSP even in the case of small internal memory. In fact, only 55.54 KB of internal memory space is required for 720p HD (1280x720) resolution. The major drawbacks of this architecture are the multiple accesses to external memory for each required to transfer a current or a reconstructed MB. It also needs to store the left and top neighboring pixels used in the prediction and filtering of the next MBs after each MB processing.

## 5.2 « MBs row level» implementation

To avoid the first architecture's drawbacks, a second implementation is proposed. The principle of this implementation as illustrated in Fig. 6 consists of loading one MBs row (16 x frame_width) from the current frame and 3 MBs rows (48 x (16+ frame_width +16)) for the search window from the reference frame to the appropriate buffers created in internal memory. The DSP core encodes the whole current MBs row without external memory access. Then, the reconstructed MBs row (20 x (16+ frame_width +16)) is transferred from LL2RAM to SDRAM memory in the reference frame. Thus, it is not necessary to create another memory buffer for the reconstructed frame. The reference frame buffer can be exploited to store the reconstructed MBs row; since overwritten data will not be used (they are already copied into the 3 MBs rows of the search window). Moving to the second current MBs row, it is not necessary to load 3 MBs rows for the search window from the reference frame, just shift up the last two MBs rows of the search window in the internal memory and bring the third from the fourth MBs row of the reference image.

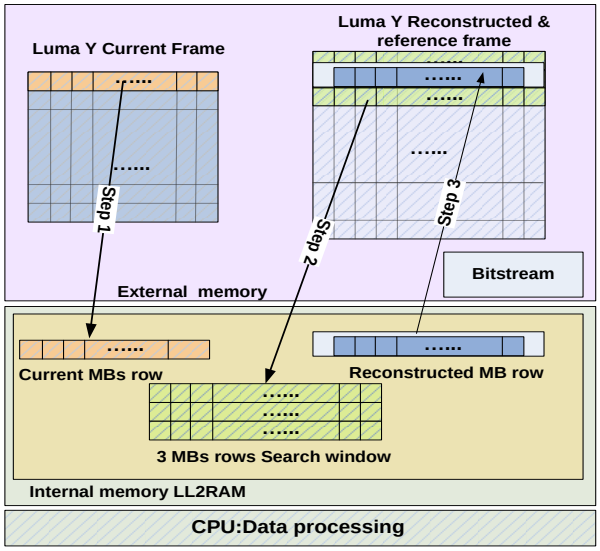


**Fig. 6** « one MBs row level » implementation

This approach outstandingly reduces the access to external memory. Thus only one memory access to external memory for reading one MB row instead of 80 accesses to read 80 MBs that form a MB row for HD 720p frame (1280/16=80). The same prevail for saving the reconstructed MB row. In addition, when proceeding at a MBs row level architecture, all the left boundaries required in the next MB prediction and filtering are already available in internal memory, so the left neighboring pixels backup is removed. Moreover, this implementation reduces the backup of TOP boundaries, since storing top boundaries is required only one time after finishing processing the whole MBs row, whereas, the MB level implementation needs to store top neighboring pixels after processing each current MB.

## 5.3 Experimental Results for the mono-core implementation

In this preliminary work, the two proposed architectures are implemented on a single DSP core TMS320C6472 running at 700 MHz using the H264/AVC LETI's codec. Experimental simulations are performed on the most commonly used video test sequences with CIF resolution downloaded from this website [33]. These sequences are a raw data in the YUV 4:2:0 format recommended by the Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG organizations [34] that developed the H264 codec. The number of processed frame is 300. The GOP size is equal to 8 as follows: IPPPPPP IPPP....P. Quantification Parameter (QP) is 30. Table 1 shows the performance of

the two implementations based on encoding speed. The second architecture can save up to 18.71% of run-time. Using a single DSP core, encoding time is yet close to real-time (25 f/s) for CIF resolution.

**Table 1** Performance evaluation of the proposed implementations for CIF (352x288) resolution

| CIF sequence | Encoding speed using MB level implementation (f/s) | Encoding speed using MB row level implementation (f/s) |
|---|---|---|
| Foreman | 19.72 | 24.19 |
| Akiyo | 20.69 | 24.73 |
| News | 21.76 | 25.21 |
| Container | 20.39 | 24.80 |
| Tb420 | 18.64 | 22.79 |
| Mobile | 18.73 | 22.77 |
| Speed average (f/s) | 19.98 | 24.58 |

After verifying that «MBs row level» architecture is the well optimized implementation, this architecture is then evaluated for higher resolutions: SD (720x480) and HD (1280x720). Table 2 presents the achieved encoding speeds when applying «MBs row level» architecture on a single DSP core for several uncompressed YUV 4:2:0 SD and HD video sequences. At first, these video sequences are downloaded in a compressed HD RGB format from the video-sharing website YouTube; then they are converted into YUV 4:2:0 format and resized into SD resolution using OpenCv library [35]. The number of processed frames is 1200 frames, QP=30 and GOP size is equal to 8.

**Table 2** Performance evaluation of the second implementations for SD and HD resolutions on a single DSP core

| sequence | Encoding speed for SD resolution on a single DSP core (f/s) | Encoding speed for HD resolution on a single DSP core (f/s) |
|---|---|---|
| Planets | 7.047 | 2.663 |
| Power of natures | 7.193 | 2.651 |
| Turtle | 6.827 | 2.609 |
| Vague | 7.03 | 2.696 |
| Nature | 7.36 | 2.756 |
| Bird | 7.928 | 2.999 |
| Speed average (f/s) | 7.23 | 2.73 |

It is clear that mono-core processors with low CPU frequency cannot meet the real-time requirement for high resolution videos. Thus, moving to a multicore implementation and exploiting the H264 parallelism are mandatory to reach real-time encoding for VGA and SD resolutions and improve the encoding speed for HD resolution.

## 6 Multicore implementation using Frame Level Parallelism

### 6.1 Implementation strategy

From previous works detailed in the past section, the conclusion that could be taken is that each of the partitioning method has as many advantages as drawbacks. 1) GOP level parallelism ensures a good speedup but involves very high encoding latency. 2) Frame level parallelism improves efficiently the encoding run-time with low latency. 3) Slice Level parallelism improves the encoding speed but induces PSNR degradation and bitrate increase. 4) Load balance, large data transfer and synchronizations between processors are the important drawbacks of MB level parallelism. 5) Regarding functional partitioning, this approach is not suitable for H.264/AVC encoder [10] due to two reasons. First, large amount of data transfer among processors will demand a large system bandwidth to assure inter-processor communication. Second, functions in H.264/AVC encoder have different load balance, so it is hard to equally map functions among processors. Thus, the final performance is always restricted by the processor with the heaviest load. Based on these observations, the frame level parallelism approach will be applied in order to enhance the encoding speed and get a low latency without inducing any rate distortion (PSNR degradation and bitrate increase).

Our multicore implementation using FLP approach will exploit the optimized mono-core architecture implemented on a single DSP core which is the «MBs row level» implementation. The approach of our real-time demo implementation is described in Fig. 7.

In a preliminary step, our DSP platform is connected to a personal computer (PC) via a Gigabit Ethernet link in order to achieve real-time TCP/IP (transmission Control Protocol /Internet Protocol) data transfers between them. The PC itself is linked to a Universal Serial Bus (USB) HD webcam to capture RAW video and send it to DSP for encoding. Once the DSP will be integrated in a smart camera system, the PC will no longer be needed.

In this work, the personal computer is used only for validation purposes because our platform has not yet a frame grabber interface. A commonly used video test sequences in YUV 4:2:0 format are used for encoding. Then, the similarity between the output of our DSP implementation and that of the PC implementation is verified. Even the Ethernet data transfer is used only to ensure a real-time data transfer but it is not our principle aim. The main of our work is to bring out the efficiency of our processor to meet real-time constraint for the most complex application (video encoding). So if this processor will be the kernel of a smart camera platform, several image and video processing applications could be performed and real-time constraint could be satisfied.

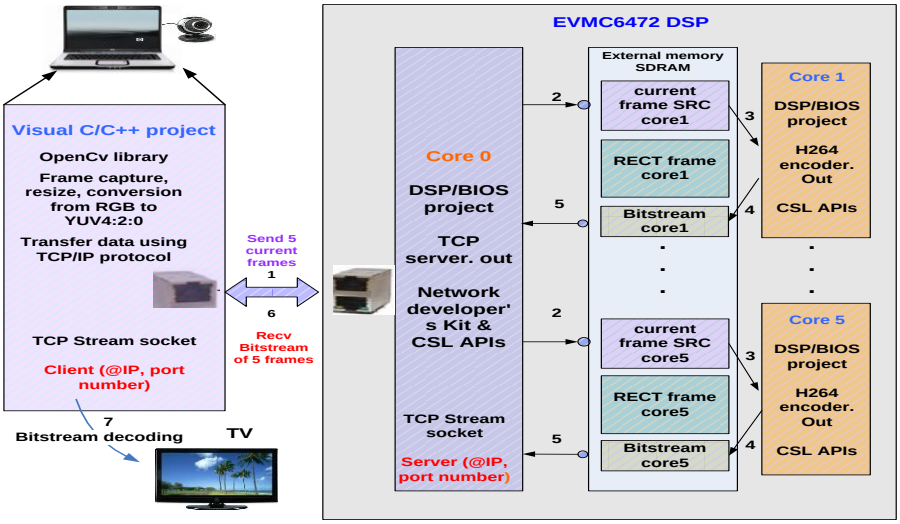


**Fig. 7** Encoding demo using Frame Level parallelism algorithm

The image acquisition will be in real-time and in this case, TCP/IP data transfers with Gigabit Ethernet will not be required since the camera interface will be directly connected with the DSP memory. This work may encourage producers to develop a new generation of smart cameras based on multicore DSPs that can perform a sophisticated image and video processing applications and capable to satisfy real-time constraint.

As our DSP platform includes 6 DSP cores, the first core "core0" is assigned as a master. It executes a TCP server program. It is devoted to establish TCP/IP connection with the client (PC) exploiting Texas Instruments (TI) NDK library (Network Developer's Kit [36]). In a first step, it receives the current frames sent by the PC after camera capture and stores them into the external memory which is a shared memory between all the DSP cores. The 5 remaining DSP cores are used to encode the 5 received frames. For each core, a memory section is reserved to store the current frame (SRC), the reconstructed frame (RECT, which will be the reference frame for the next core) and finally a bitstream buffer where the bitstream will be stored. After encoding, the core0 server sends the bitstream of all encoded frames to the client (PC) in order to store or display it. Into the internal program memory of core0, a TCP server program is loaded to establish connection between the DSP and the PC. H264/AVC algorithm is loaded into each internal program memory of the 5 remaining cores. Thus, a C++ project is developed and executed on the PC in order to capture video from the camera. Our program is based on OpenCv library which is used to convert the captured frames from RGB to $YC_rC_b$ 4:2:0 format. A TCP socket (@IP, Port number) is created to transmit data between core0 (server) and the PC (client).

When applying frame level parallelism and exploiting «One MBs row level implementation», core i starts encoding its appropriate frame only if core i-1 has finished encoding at least 3 MBs rows from the previous frame. These 3 MBs rows will be used as the search window for the motion estimation of the first MBs row of the current frame processed by core i (see section 5.2). Thus, inter data dependency is respected and consequently, no rate distortion will be provided.

The steps of encoding a video sequence using FLP are detailed as follows (Cf. Fig. 8):

- After establishing connection between the PC and the DSP, core0 receives 5 frames from the PC as 5 cores are devoted to encoding. Each frame is loaded into the SRC buffer of each remaining core (1-5).
- When the reception of the 5 current frames is completed, core0 sends 5 inter processor communication interruption events (IPC) to cores 1-5; which are in a wait state for an interruption event from core0; to indicate that SRC frames are already in external memory so they can start encoding.
- Core1 is the first core that begins encoding. Upon completion encoding the first 3 MBs rows of the SRC frame, it sends an IPC to the next core (core2) which itself is in a wait state for an interruption from core1 to start encoding its appropriate frame. The same procedure will be reproduced from core3 to core5.
- To avoid that core *i* exceeds core *i-1* (which is possible because the load balance is not uniform between successive frames and it could give an erroneous result), the encoding of the next MBs row is conditioned with the reception of an IPC from the previous core. Thus, each core will send an IPC to its next core after encoding a MBs row that its index is higher than 3. Since each core starts encoding after its previous core finishes encoding 3 MBs rows, it should not wait an IPC from the previous core to encode the last 2 MBs rows of each SRC frame; otherwise encoding will be blocked by waiting an incoming IPC. As a result, each core will totally send Max_MBs_rows - 2 interruptions to the next core. When all cores finish encoding the current frames and specifically core5 which is the last core that finishes its task, cores1 to 5 send 5 IPCs to core0 which is in a wait state to indicate that the bitstream of 5 frames is ready in external memory to be transferred to the PC.

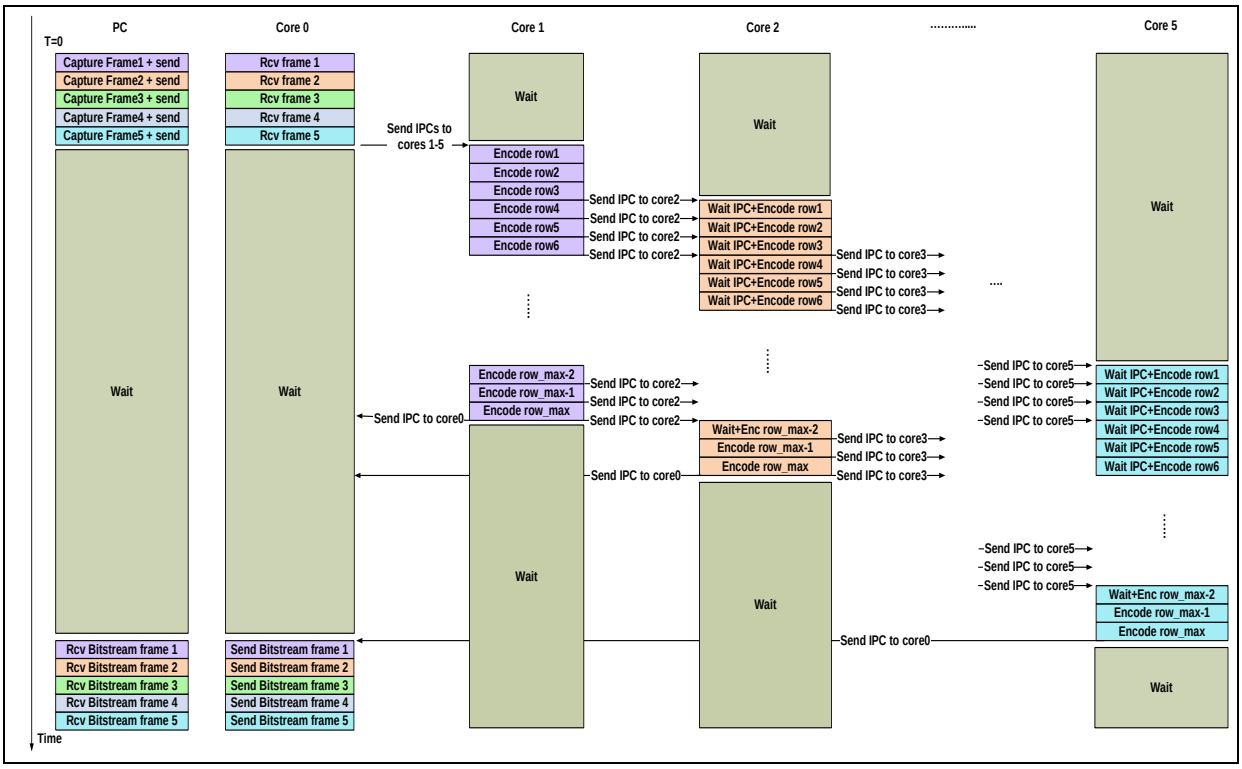


**Fig. 8** The chronological steps of the Frame Level Parallelism approach on the multicore DSP TMS320C6472

- When receiving these 5 IPCs, core0 sends the bitstream of the 5 frames to PC via the Gigabit Ethernet link.
- After the end of bitstream receiving, the PC captures another 5 frames and sends them to core0. The same work thereby will be reproduced.

## 6.2 Cache coherency

Multicore processing often leads to cache coherency problem. This is due to the simultaneous access of two or more cores with a separate cache memory for each core to the same location in a shared memory. In general purpose multiprocessor, programmers don't have such problem because it is controlled automatically by a complex hardware. But in our multicore DSP architecture, designers have to control it, since there is no such automatic controller. In order to deal with cache coherency, the Chip Support Library (CSL library) [37] from TI provides two API commands:

- CACHE_wbL2((void *)XmtBuf, bytecount, CACHE_WAIT) to write back the cached data from the cache memory to its location in the shared memory.
- CACHE_invL2((void *)RcvBuf, bytecount, CACHE_WAIT) to invalidate the cache lines and force the CPU to read data from its location in the shared memory.

In our case, when core0 receives the current frames from the PC, it should write back the cached data to external memory. In the other side, core1 to core5 should invalidate the current SRC frames addresses in the cache memory before starting encoding in order to use the updated data. Also, when core1 to core5 complete encoding, they should write back the bitstreams from the cache memory to the external memory in order to overcome the cache coherence with core0 which will send the bitstream from external memory to the PC. Furthermore, among core1 to core5, the problem of cache coherency exists because core *i* will read data (the search window) written by core *i-1* (Reconstructed MBs row). So, the same principle should be applied. Before sending an IPC to the next core, a write-back of the reconstructed MBs row must be applied. In the other side, the next core should invalidate the cached data of the search window before starting encoding in order to use an updated data written by the previous core.

## 6.3 Experimental results for the Frame Level Parallelism implementation on 5 DSP cores

When applying the Frame Level Parallelism method with the « One MBs row level architecture» on 5 DSP cores, each core is delayed by 3 MBs row from its antecedent. Thus, the fifth core is delayed by 12 MBs rows with respect to the first core. Let consider $T$ the average time needed to encode a MBs row and Max_MBs_row the number of MBs rows in a frame equal to the frame's height divided by the MB's height. So, encoding 5 frames using FLP approach on 5 DSP cores needs Max_MBs_row*$T$+4*3*$T$ instead of 5* Max_MBs_row*$T$ for sequential encoding. Thus, the different theoretical speedup factors that could be reached for different resolution are computed as follows:

For CIF (352x288), Max_MBs_row=288/16=18 and $speedup = \frac{(18x5)xT}{(12+18)xT} = 3$.
For SD (720x480), Max_MBs_row=480/16=30 and $speedup = \frac{(30x5)xT}{(12+30)xT} = 3.57$.
For HD (1280x720), Max_MBs_row=720/16=45 and $speedup = \frac{(45x5)xT}{(12+45)xT} = 3.94$.

Several experiments have been performed on the same video sequences used in the mono-core implementation in order to correctly evaluate the performance of the two implementations. Tables 3, 4 and 5 illustrate respectively the encoding speeds (f/s) for CIF, SD and HD resolutions for the mono-core and the multicore implementations. 5 DSP cores, running each at 700 MHz, are exploited for H264/AVC encoding using the FLP approach presented above. The speedup is also computed and presented for each video sequence.

The number of frames to be encoded is 300 frames for CIF resolution and 1200 frames for SD and HD resolution. The chosen QP is 30 and GOP size is 8 (IPPPPPP IPPP…).

When using 5 cores with GOP size equal to 8, the intra frame "I" will be firstly processed by the core1 then by the core4 for the second GOP then by the core2 for the third GOP etc. So, core1 does not process only "I" frames but also a "P" frames and in this case, its reference frame is the reconstructed frame of the last core which is the core5. If GOP size is equal to 5 (IPPPP IPPPP…) core1 in this case will process only intra frames and as result, load balance is not uniform among DSP cores.

Experiments on 5 DSP cores show that speedup factors of 2.92, 3.33 and 3.74 are achieved respectively for CIF, SD and HD resolutions. Experimental results approximately verify the theoretical results. Thus, the obtained speedup factors are lightly less than the maximal speedups. This is due to: inter-communications needed among different cores, write-backs and cached data invalidations. The proposed FLP implementation achieves an encoding speed about 70 f/s for CIF resolution surpassing real-time constraint of 25 f/s. Encoding speed is efficiently improved for SD and HD resolutions compared to mono-core implementation.

Encoding speed for SD resolution is very close to real-time since the average encoding speed is 24 f/s.

**Table 3** Encoding speed for CIF (352x288) resolution

| CIF sequence | Encoding speed on one core (f/s) | Encoding speed on 5 cores (f/s) | Speedup |
|---|---|---|---|
| Foreman | 24.19 | 71.22 | 2.94 |
| Akiyo | 24.73 | 72.36 | 2.93 |
| News | 25.21 | 74.16 | 2.94 |
| Container | 24.80 | 72.18 | 2.91 |
| Tb420 | 22.79 | 65.66 | 2.88 |
| Mobile | 22.77 | 66.99 | 2.94 |
| Average | 24.58 | 70.43 | 2.92 |

**Table 4** Encoding speed for SD (720x480) resolution

| SD sequence | Encoding speed on one core (f/s) | Encoding speed on 5 cores (f/s) | Speedup |
|---|---|---|---|
| Planets | 7.047 | 23.76 | 3.37 |
| Power of natures | 7.193 | 23.58 | 3.27 |
| Turtle | 6.827 | 23.24 | 3.40 |
| Vague | 7.03 | 24.11 | 3.43 |
| Nature | 7.36 | 23.63 | 3.21 |
| Bird | 7.928 | 26.24 | 3.31 |
| Average | 7.23 | 24.09 | 3.33 |

**Table 5** Encoding speed for HD (1280x720) resolution

| HD sequence | Encoding speed on one core (f/s) | Encoding speed on 5 cores (f/s) | Speedup |
|---|---|---|---|
| Planets | 2.663 | 10.12 | 3.80 |
| Power of natures | 2.651 | 9.78 | 3.70 |
| Turtle | 2.609 | 9.96 | 3.82 |
| Vague | 2.696 | 10.20 | 3.78 |
| Nature | 2.756 | 9.97 | 3.62 |
| Bird | 2.999 | 11.09 | 3.70 |
| Average | 2.73 | 10.18 | 3.74 |

## 7 Enhanced Frame Level Parallelism approach: hiding communication overhead

The first implementation of the FLP approach improves the encoding speed compared to the mono-core implementation but does not efficiently exploit the DSP cores. A lot of time is wasted (processor waiting data) which reduces our multicore implementation efficiency. Moreover, communication overhead is not optimized. To avoid these drawbacks, this part presents the enhanced version of FLP approach based on hiding communication overhead. For the first version of the FLP approach, core1 to core5 wait that core0 completes the reception of 5 frames, although encoding can be immediately started after the reception of the first frame. Furthermore, core0 waits that core1 to core5 finish encoding their respective frames in order to start sending the bitstreams, although it can start sending to the PC any available bitstream. In the other side also, during encoding, core0 is in a wait state; so this time could be exploited to prepare the next 5 frames in order to overlap frames encoding and frames reading processes. To realize these optimizations, a ping pong buffer is used for each SRC frame instead of a single buffer used for the first implementation as shown in Fig. 9. A multithreading approach is employed on the PC side. Three threads are used to manage reading raw frames, sending them via Ethernet, receiving encoded bitstream and saving it in a file.

The strategy of our implementation is described in Fig. 10 and consists of the following steps:

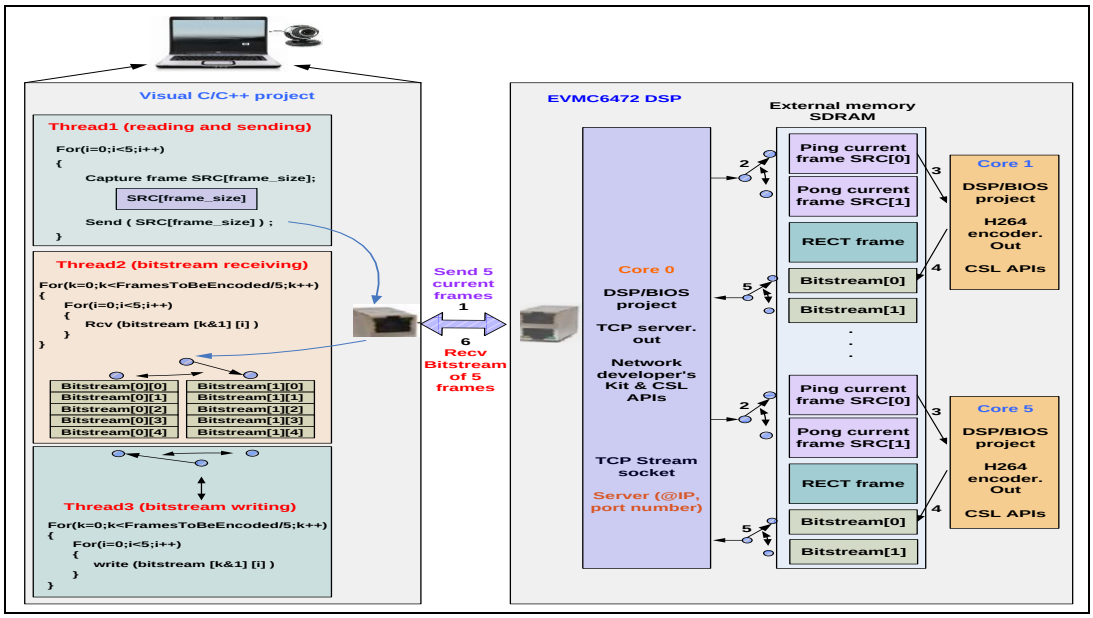


**Fig. 9** The enhanced Frame Level Parallelism approach

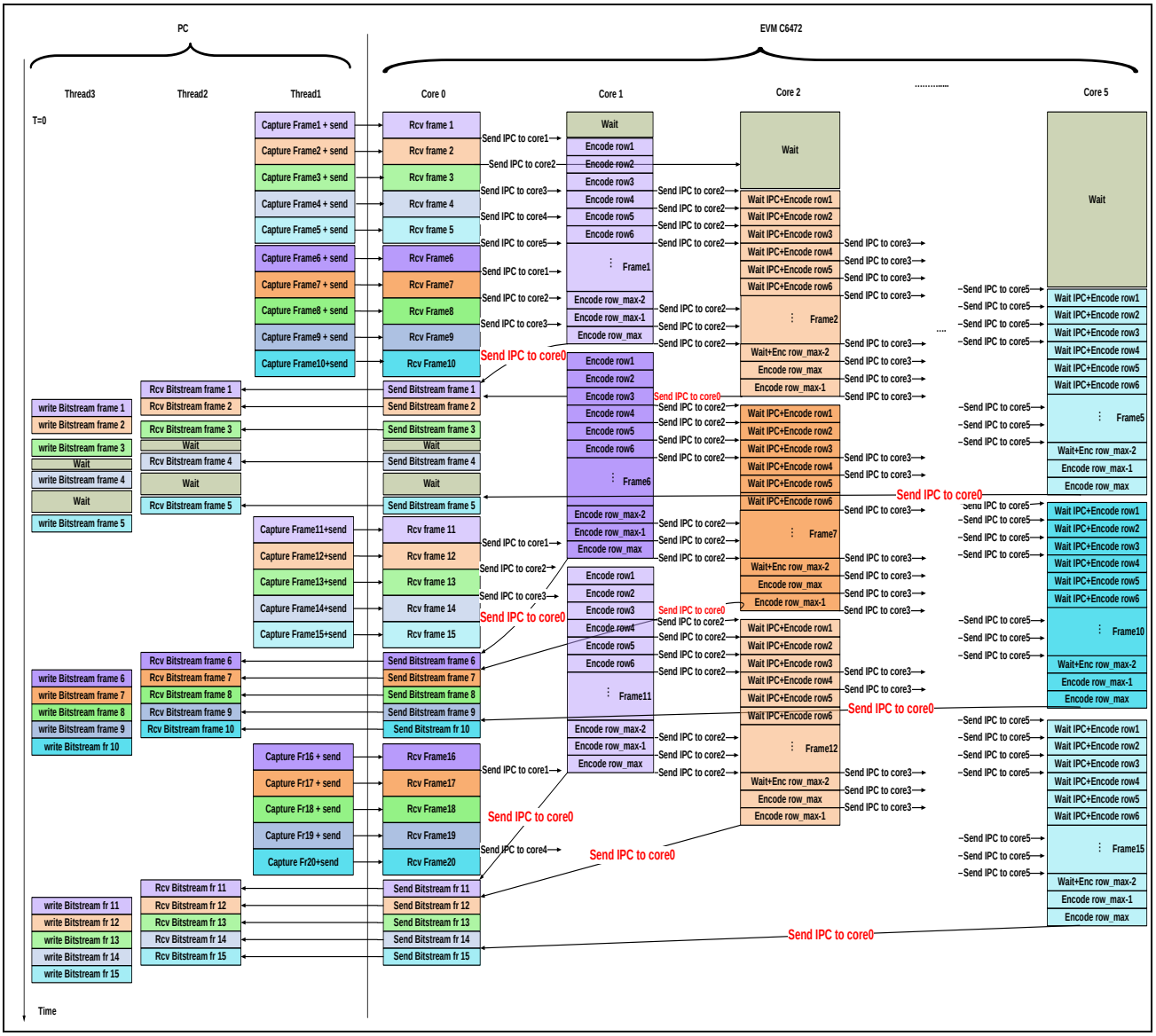


**Fig. 10** The chronological steps of Enhanced Frame Level Parallelism on the multicore DSP TMS320C6472

- The first thread "thread1" captures the first frame from the camera and sends it to core0 which will store it into the ping buffer SRC[0] of core1. Core0 sends then an IPC to core1 to indicate that it can start encoding its current frame.
- When receiving an IPC from core0, core1 triggers the encoding. At the same time thread1 moves to read and send the second frame to core0 which will store it into the ping buffer of core2. This step is repeated until receiving of 5 frames. Thus, each core immediately starts encoding after core0 receives its current frame without waiting the reception of all the frames.
- While core1 to core5 encode their frames with the same principle as the first FLP implementation, thread1 sends the next 5 frames to core0 which will store them into the pong buffers SRC[1] for 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 encoding is achieved on a core *i*, the bitstream is stored into the ping buffer bitstream[0]. Then, core *i* sends an IPC to core0 to inform that it can forward its bitstream to the PC. After that, core *i* starts encoding its pong frame stored into SRC[1] without any wait and stores the bitstream into the pong buffer bitstream[1] (to not overwrite data stored into bitstream[0]).
- 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. The second thread "thread2" receives the ping bitsteams and stores them into the ping buffers Bitstream[0][i]. Then, the third thread "thread3" writes the bitstreams in a file and at the same time thread1 sends the next 5 frames to core0 which will store them into the ping buffers SRC[0] of each core. With this technique, the ping bitstreams writing, the pong SRC frames encoding and the next 5 ping SRC frames capturing and sending are processed in parallel.
- The processing is then looped in a reverse order for SRC frames and bitstreams through ping pong buffers.

When looking at Fig. 10, no significant delays have occurred. All cores process their respective data without any waiting time. The enhanced FLP algorithm efficiently exploits the multicore platform. Multithreading algorithm with ping pong buffers technique efficiently overlap data transfer with encoding process.

### 7.1 Experimental results for the Enhanced FLP implementation on 5 DSP cores

To evaluate our enhanced FLP approach implementation in terms of encoding speed and speedup factor, several experiments have been performed on different video sequences with different resolutions as the first implementations.

Table 6, 7 and 8 show respectively the achieved encoding speeds and speedup factors for the two implementations: the mono-core implementation and the enhanced FLP implementation on 5 DSP cores. For SD and HD resolutions, 1200 frames are encoded and 300 frames for CIF resolution. The used QP is equal to 30 and GOP size is 8. The presented results prove that our enhanced FLP implementation allows us to meet the real-time constraint for CIF and SD resolutions. Our encoder can process up to 98 f/s for CIF sequences and 31 f/s for SD resolution. Experiments show that a speedup of more than 4 times is achieved (4.11 for CIF, 4.38 for SD and 4.52 for HD). The real-time is not yet achieved for HD resolution but our enhanced FLP allows us to save up to 77% of encoding time and processes up to 12 f/s instead of 2.73 f/s on a single core.

**Table 6** Encoding speed for CIF (352x288) resolution between the mono-core and the enhanced FLP implementation

| CIF sequence | Encoding speed on a single core (f/s) | Encoding speed on 5 DSP cores (f/s) | Speedup |
|---|---|---|---|
| Foreman | 24.19 | 99.55 | 4.11 |
| Akiyo | 24.73 | 102.08 | 4.13 |
| News | 25.21 | 103.77 | 4.12 |
| Container | 24.80 | 102.19 | 4.12 |
| Tb420 | 22.79 | 94.14 | 4.13 |
| Mobile | 22.77 | 91.20 | 4.00 |
| Average | 24.58 | 98.82 | 4.11 |

**Table 7** Encoding speed for SD (720x480) resolution between the mono-core and the enhanced FLP implementation

| SD sequence | Encoding speed on a single core (f/s) | Encoding speed on 5 DSP cores (f/s) | Speedup |
|---|---|---|---|
| Planets | 7.047 | 30.70 | 4.36 |
| Power of natures | 7.193 | 31.26 | 4.34 |
| Tortue | 6.827 | 30.58 | 4.48 |
| Vague | 7.03 | 30.83 | 4.38 |
| Nature | 7.36 | 32.22 | 4.38 |
| Bird | 7.928 | 34.83 | 4.39 |
| Average | 7.23 | 31.73 | 4.38 |

**Table 8** Encoding speed for HD (1280x720) resolution between the mono-core and the Enhanced FLP implementation

| HD sequence | Encoding speed on a single core (f/s) | Encoding speed on 5 DSP cores (f/s) | Speedup |
|---|---|---|---|
| Planets | 2.663 | 12.03 | 4.52 |
| Power of natures | 2.651 | 11.93 | 4.50 |
| Tortue | 2.609 | 11.71 | 4.49 |
| Vague | 2.696 | 12.23 | 4.54 |
| Nature | 2.756 | 12.43 | 4.51 |
| Bird | 2.999 | 13.59 | 4.53 |
| Average | 2.73 | 12.32 | 4.52 |

During our measure of the enhanced FLP encoding speed, the cost of data transfer is taken into account. The time of capturing frames, transferring them to DSP, receiving them by core0, and loading them to DSP memory has consequently been added to the encoding time in order to evaluate the efficiency of our enhancement.

Experimental results show that our proposed data transfer scheduling technique completely hides the

communication overhead. The time of data transfers does not contribute in the run-time thanks to using the ping pong buffer technique and multi-threading processing. The achieved speedup factor is higher than the non-optimized FLP since the obtained speedup factor is 4.52x instead of 3.74x for HD resolution. Encoding speed is significantly increased from 24 f/s to 31.73 f/s surpassing the real-time compliant for SD resolution. Our proposed enhancement efficiently exploits our multicore platform and allows us to get a good saving time (77%) for HD resolution.

Finally, for low and medium video resolutions such as CIF, VGA (640x480) and SD, some cores could be free which allowing us to exploit them to perform other tasks (biometric recognition and access control, texture and position detection, surveillance application etc). This will give an important advantage for our multicore DSP if integrated into a smart camera system.

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 causes a significant latency.

First, our encoding implementation is based on "MB row level architecture", so each core reads a MB row from the external memory to the internal L2 memory. The processing will be performed thereafter 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 cache by the cache controller. This allows reducing the cache misses so accelerating encoding run-time. Reconstructed fraction and bitstream are not copied directly into the external memory after their processing but they are kept into 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 C6472 includes also a switch fabric module that provides arbitration between the different cores to access the external memory which is a 32-bit DDR2-533 SDRAM with up to 2133 MBps of throughput. Several test show that this bandwidth is enough to support multiple DSP cores accessing the DDR2 memory simultaneously [38]. Finally, the DDR2 memory on the C6472 EVM contains eight banks and every bank can have an open row or page, so eight rows can be opened at the same time. This dramatically reduces the row switch overhead.

Table 9 presents a comparison between our approach and other implementations performed on several platforms and applying different methods of parallelism. Experiments show that 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 performance.

**Table 9** Comparison of parallelization approaches on different platforms

| approach | Our approach | [9] | [10] | [12] | [21] | [23] | [25] |
|---|---|---|---|---|---|---|---|
| Partitioning method | Frame | Task | MB region partition (MBRP) | MB/Frame | slice | GOP | Task |
| platform | Multicore DSP TMS320C6472 (5 cores for encoding) | 167-core asynchronous array of simple processors | PC with a P4 1.7GHz processor 4 cores | Pentium 4 processor running at 2.8 GHz | quad TMS320C6201 DSP system | 3 Microblaze soft cores based on XILINX FPGA | 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. | 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. | 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. | AVS reference code RM5.2, ME algorithm is full search, 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) | 98 f/s for CIF, 32 f/s for SD and 12 f/s for HD | 21 f/s for VGA (640 x 480) | 0.6 f/s for CIF and 0.15 f/s for SD | 0.58 f/s for CIF | 30 f/s only for CIF resolution | 3 f/s for QCIF | 30 f/s for HD720p |
| Distortion PSNR/bitrate | No | yes | No | No | yes | No | Yes |

Other works achieve real-time processing for low resolution but not yet for higher resolutions. GPU's implementation allows achieving real-time for HD resolution thanks to the great number of processing cores but in the other side, this proposed scheme induces some rate distortion (PSNR degradation and bitrate increment). GPU platform remains an interesting solution to meet real-time requirement for high computational applications but it is not intended for embedded systems that require low power consumption. Our implementation ensures a good encoding scalability without inducing any rate distortion.

## 8 Conclusion

In this paper, an optimized implementation of the H264/AVC encoder on a multicore DSP TMS320C6472 was presented. The Frame Level parallelism approach was used to accelerate encoding speed. Hiding communication overhead allowed enhancing the FLP implementation and improving the speedup factors. Experiments of enhanced FLP on 5 DSP cores running at 700 MHz showed that real-time was achieved by reaching 98 f/s for CIF resolution and 32 f/s for SD resolution as encoding speeds. Our parallel implementation saved up to 77% of encoding time for HD resolution and ensured a good encoding speedup factors ranging from 4.11 to 4.52 without providing any quality degradation or bitrate increase. Our work validated the capability of real-time processing, even for high complexity applications, by smart camera systems if they are based on embedded multicore DSP. As perspectives, we will try to reach real-time encoding for HD resolution by implementing our approach on the latest generation of Texas Instruments DSP (TMS320C6678). It includes 8 DSP cores each running at 1.25 GHz, giving a large possibility to achieve real-time constraint for HD resolution. Also, two partitioning methods could be combined in order to improve encoding efficiency. Power consumption of our multicore implementation will be taken into account to more evaluate our embedded encoder. All this work will be reusable to implement the new HEVC-H265 video standard. This knowledge will be use for our next task: H265 real-time implementation on the TMS320C6678 DSP.


## Acknowledgements

This work is fruit of cooperation between Sfax National School of Engineers and ESIEE Engineering PARIS. It is supported 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] Smart camera [Online].Available: http://en.wikipedia.org/wiki/Smart_camera

[2] Yu Shi, Serge Lichman, "Smart Cameras: A Review," [Online].Available: http://www.nicta.com.au/__data/assets/pdf_file/0004/16456/smart_camera_review.pdf

[3] 8-Megapixel OEM Smart Camera Module with Removable Storage [Online].Available: http://www.mosaicengineering.com/docs/OEM-SCM.2010-01-26.pdf

[4] Smart Camera manufacturers and products links [Online].Available: http://www.smartcamera.it/links.htm#Top

[5] Freescale Machine Vision and Camera[Online].Available: http://www.freescale.com/webapp/sps/site/application.jsp?code=APLMAVISMCA

[6] TMS320C66x high-performance multicore DSPs for video surveillance [Online].Available: http://www.ti.com/lit/ml/sprt643/sprt643.pdf

[7] Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG, "Draft ITU-T Recommendation and Final Draft international Standard of Joint Video Specification (ITU-T Rec. H.264 ISO/IEC 14496-10 AVC)", JVT-G050, 2003.

[8] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand, "Overview of the High Efficiency Video Coding (HEVC) Standard", IEEE Trans. Circuits and Systems for Video Technology, Vol. 22, No. 12, pp. 1649-1668, Dec. 2012.

[9] Zhibin Xiao; Le, S.; Baas, B., "A fine-grained parallel implementation of a H.264/AVC encoder on a 167-processor computational platform," Signals, Systems and Computers (ASILOMAR), 2011 Conference Record of the Forty Fifth Asilomar Conference on , vol., no., pp.2067,2071, 6-9 Nov. 2011. doi: 10.1109/ACSSC.2011.6190391

[10] 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

[11] H264/AVC software Joint Model JM: http://iphome.hhi.de/suehring/tml/download/old_jm/

[12] 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. doi: 10.1109/ICASSP.2006.1661319

[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 doi: 10.1109/IPDPS.2004.1302990

[14] 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 Simulation(ICCSS 2011), IPCSIT vol. 7, pp.127-132, May 2011, IACSIT Press, Singapore, ISSN:2010-460X, ISBN : 978-981-08-8638-7,Bangkok,27-28,May 2011.

[15] 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.

[16] 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. doi: 10.1109/PARELEC.2006.42.

[17] 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. doi: 10.1109/HPCC.2010.82.

[18] 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. doi: 10.1109/ICME.2009.5202703

[19] Zrida, H.K.; Jemai, A.; Ammari, A.C.; Abid, M., "High level H.264/AVC video encoder parallelization for multiprocessor implementation," Design, Automation & Test in Europe Conference & Exhibition, 2009. DATE '09. , vol., no., pp.940,945, 20-24 April 2009. doi: 10.1109/DATE.2009.5090800

[20] Antonio Rodrigues, Nuno Roma, and Leonel Sousa. 2010. p264: open platform for designing parallel H.264/AVC video encoders on multi-core systems. In Proceedings of the 20th international workshop on Network and operating systems support for digital audio and video (NOSSDAV '10). ACM, New York, NY, USA, 81-86. DOI=10.1145/1806565.1806586 http://doi.acm.org/10.1145/1806565.1806586

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

[22] Bruno Alexandre de Medeiros, "Video coding on multicore graphics processors (GPUs)," Dissertation submitted to obtain the Master Degree in Information Systems and Computer Engineering, High Institute of Techniques, Technical University of Lisboa, November 2012.

[23] 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 doi: 10.1109/ICCIS.2013.400

[24] Xiulian Peng; Jizheng Xu; You Zhou; Feng Wu, "Highly Parallel Line-Based Image Coding for Many Cores," Image Processing, IEEE Transactions on , vol.21, no.1, pp.196,206, Jan. 2012. doi: 10.1109/TIP.2011.2159986

[25] 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. doi:10.1155/2014/716020

[26] A. O. Adeyemi-Ejeye and S. Walker, "4kUHD H264 Wireless Live Video Streaming Using CUDA," Journal of Electrical and Computer Engineering, vol. 2014, Article ID 183716, 12 pages, 2014. doi:10.1155/2014/183716

[27] Wajdi Elhamzi, Julien Dubois, Johel Miteran, Mohamed Atri, Barthelemy Heyrman, Dominique Ginhac, Efficient smart-camera accelerator: A configurable motion estimator dedicated to video codec, Journal of Systems Architecture, Volume 59, Issue 10, Part A, November 2013, Pages 870-877, ISSN 1383-7621

[28] TMS320C6472 datasheet, [Online].Available: http://www.ti.com/lit/ds/sprs612g/sprs612g.pdf

[29] Multicore DSP vs GPUs, [Online].Available: http://www.sagivtech.com/contentManagment/uploadedFiles/fileGallery/Multi_core_DSPs_vs_GPUs_TI_for_distribution.pdf

[30] TMS32C6472 low power consuption, [Online].Available: http://www.ti.com/lit/wp/spry130/spry130.pdf

[31] Werda, I.; Dammak, T.; Grandpierre, T.; Ayed, M. & Masmoudi, N.; "Real-time H.264/AVC baseline decoder implementation on TMS320C6416," Journal of Real-Time Image Processing, Springer-Verlag, 2012, 7, 215-232, 2012.

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

[33] Xiph.org Video Test Media Online].Available: https://media.xiph.org/video/derf/

[34] 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

[35] Open source computer vision library [Online].Available: http://opencv.org/

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

[37] TMS320C6472 Chip Support Library API reference Guide [Online].Available: http://software-dl.ti.com/sdoemb/sdoemb_public_sw/csl/CSL_C6472/latest/index_FDS.html

[38] Throughput Application Report for the TMS320C6472 DSP [online]. Available: http://www.ti.com/lit/an/spraay0a/spraay0a.pdf