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

# High Level Optimized Parallel Specification of a H.264/AVC Video Encoder

**Hajer Krichene Zrida[1], Ahmed C. Ammari[2], Abderrazek Jemai[3], Mohamed Abid[1]**

[1] Department of Electrical Engineering, Computer and Embedded Systems Laboratory, ENIS Institute, Sfax University, Tunisia
[2]Unité de recherche en Matériaux Mesures et Applications (MMA) INSAT BP676 - 1080 Tunis CEDEX Tunisie
[3]Laboratoire LIP2-Faculté des Sciences de Tunis 1060 Belvédère Tunis, Tunisie

**Abstract:** *H.264/AVC (Advanced Video Codec) is a new video coding standard developed by a joint effort of the ITU-TVCEG and ISO/IEC MPEG. This standard provides higher coding efficiency relative to former standards at the expense of higher computational requirements. Implementing the H.264 video encoder for an embedded System-on-Chip (SoC) is thus a big challenge. For an efficient implementation, we motivate the use of multiprocessor platforms for the execution of a parallel model of the encoder. For this purpose, we proposed a high-level parallelization approach for the development of an optimized parallel model of a H.264/AVC encoder for embedded SoCs. This approach is used independently of the architectural issues of any target platform. It is based on the exploration of the task and data levels forms of parallelism simultaneously, and the use of the parallel Kahn process network (KPN) model of computation and the YAPI programming C++ runtime library. To demonstrate the effectiveness of the obtained parallel model of the H.264 encoder, the encoding performances have been evaluated by system-level simulations targeting multiple multiprocessors platforms.*

**Keywords:** *The H.264/AVC Video Encoder, Task and Data levels parallelization, Kahn Process Networks model of computation and YAPI parallel programming library, System-level Performance Evaluation.*

***Received:*** *September 05, 2010* | ***Revised:*** *December 31, 2010* | ***Accepted:*** *February 05, 2011*

## 1. Introduction

The H.264/AVC has been designed with the goal of enabling significantly improved compression performance relative to all existing video coding standards [1]. Such a standard uses advanced compression techniques that in turn, require high computational power [2]. For a H.264/AVC encoder using all the new coding features, more than 50% average bit saving with 1–2 dB PSNR video quality gain are achieved compared to previous video encoding standards [3]. However, this comes with a complexity increase of a factor 2 for the decoder and larger than one order of magnitude for the encoder [3]. Implementing a H.264/AVC video encoder represents a big challenge for resource-constrained multimedia systems such as wireless devices or high-volume consumer electronics since this requires very high computational power to achieve real-time encoding. Actually, using a single processor to real time encode H.264 bit streams may require a high performance, high frequency super scalar processor. Such a choice is not suitable for embedded systems that have strict power and cost constraints. For such a case, it may be probably necessary to use some kind of multiprocessor approach to share the encoding application execution time between several processors.

Prior to the multiprocessor implementation, the sequential H.264/AVC reference code has to be first distributed using appropriate high level parallel programming model of computation. To do so, several multiprocessor and multi-threading encoding systems and parallel implementation methodologies have been proposed and discussed in many previous studies [4, 5, 6, 7, 8 and 9]. Based on the performance results obtained in these previous works, and given our concern with resource constrained devices, we developed in a previous work a new high-level independent target-architecture parallelization approach [10 and 11] based on the parallel programming models of computation and simultaneous exploration of the two predominant concepts of parallelism; the data-level partitioning and the task-level splitting and merging. The goal of this approach is to derive in a structured way a

parallel model of the encoder with the best computation and communication workload balance.

After a brief presentation in section 2 of the main innovations of the H.264/AVC standard along with its performance and complexity analysis, this paper first discusses in section 3 the main video standards previous parallelization studies, and then reviews the key characteristics of our proposed parallelization approach. Based on this parallelization approach, a starting parallel model of the H.264/AVC reference encoder is proposed in section 4. The implementation of this model is performed according to an appropriate programming strategy. Given the high complexity of the H.264/AVC standard, details about implementation issues are discussed to cope with related memory management problems. According to the communication and computation concurrency properties of the implemented starting model, section 5 of the paper will consider concurrency optimizations using task-merging and data-partitioning forms of parallelism. This will lead to an optimized parallel model with the best computation and communication workload balance. Performance evaluation of this model targeting multiple multiprocessors platforms is discussed in section 6 and will demonstrate the effectiveness of the proposed parallel model in comparison to previous parallel implementations of the standard.

## 2. Overview of the H.264/AVC video encoder

An important concept in the design of H.264/AVC is the separation of the standard into two distinct layers: a video coding layer (VCL), which is responsible for generating an efficient representation of the video data; and a network adaptation layer (NAL) [1] which is responsible for packaging the coded data in an appropriate manner based on the characteristics of the network upon which the data will be used. This paper is concerned with the VCL layer.

### 2.1 The Coding layer Block diagram

The block diagram of the video coding layer of a H.264/AVC encoder is presented in figure1. This figure includes a forward path (left to right) and a reconstruction path (right to left) [1].

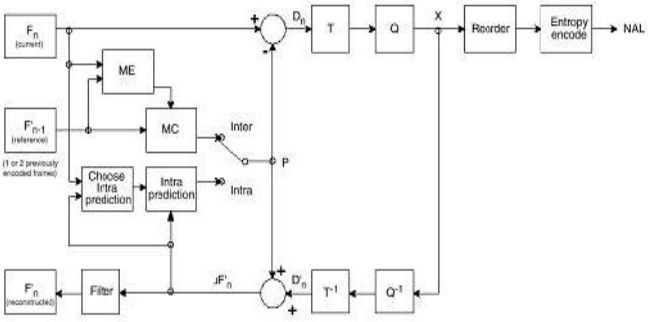


**Figure1.** H.264 /AVC video encoder block diagram

An input frame or field $F_n$ is processed in units of a macro-block (MB). Each MB is encoded in intra or inter mode and, for each block in the MB, a prediction PRED (marked 'P' in figure1) is formed based on reconstructed picture samples. In Intra mode, PRED is formed from spatially neighboring samples in the current slice that have previously been encoded, decoded and reconstructed ($uF'_n$ in the figure1 note that unfiltered samples are used to form PRED). The encoding process chooses which and how neighboring samples are used for Intra prediction, which is simultaneously conducted at the encoder and decoder using the transmitted Intra prediction side information [12].

In Inter mode, PRED is formed by motion-compensated prediction from one or multiple reference picture(s) selected from the set of reference pictures. In the figure1, the reference picture is shown as the previous encoded picture $F'_{n-1}$ but the prediction reference for each MB partition (in inter mode) may be chosen from a selection of past or future pictures (in display order) that have already been encoded, reconstructed and filtered. The prediction PRED is subtracted from the current block to produce a residual difference block $D_n$ that is transformed (using a block transform) and quantized to give X, a set of quantized transform coefficients which are reordered and entropy encoded. The entropy-encoded coefficients, together with side information required to decode each block within the MB (prediction modes, quantization parameter, motion vector information, etc.) form the compressed bit stream which is passed to a Network Abstraction Layer (NAL) for transmission or storage.

As well as encoding and transmitting each block in a MB, the encoder decodes (reconstructs) it to provide a reference for further predictions. The coefficients X are scaled ($Q^{-1}$) and inverse transformed ($T^{-1}$) to produce a difference block $D'_n$. The prediction block PRED is added to $D'_n$ to create a reconstructed block $uF'_n$ a decoded version of the original block (u indicates that it is unfiltered). A filter is applied to reduce the effects of blocking distortion and the reconstructed reference picture is created from a series of blocks $F'_n$.

### 2.2 Main innovations in comparison to previous standards

The basic functional elements of H.264/AVC presented in figure1 represent a similar set of the generic DPCM/DCT [1] coding and decoding functions of earlier standards. The H.264 provides higher coding efficiency through added features and functionality that in turn entails additional complexity. Many previous studies have presented a summary of the most relevant key features for the performance of this standard [12], [13], [14], [15]. These features concern the motion compensation model, the intra-frame prediction, the concept of Bi-predictive (B)

slices, the T transform and the entropy coding methods.

Here we present a summary of the most relevant key features for the motion compensation model. This model supports the use of multiple reference frames for prediction with a weighted combination of the prediction signals. Also, it introduces variable block-size motion compensation with small block sizes that range from 16x16 up to 7 modes including 16x8, 8x16, 8x8, 8x4, 4x8 and 4x4 pixel blocks. Motion vectors can be specified with higher spatial accuracy with quarter-pixel and eighth-pixel instead of half-pixel accuracy. In order to estimate and compensate fractional displacements, the image signal of the reference image has to be generated on sub positions by interpolation. Pixel interpolation is based on a finite impulse response (FIR) filtering operation: 6 taps for the quarter resolution and 8 taps for the eighth one [12]. A rate-distortion (RD) Lagrangian technique optimizes both motion estimation and coding mode decisions. Moreover, an adaptive deblocking filter is added to reduce visual artifacts produced by the block-based structure of the coding process [11]. For the intra-frame prediction, in contrast to previous video coding standards where prediction is conducted in the transform domain, prediction in H.264/AVC is always conducted in the spatial domain by referring to neighboring samples of already coded blocks [13]. Two classes of intra coding modes are supported. When using the INTRA-4x4 classes, each 4x4 block utilizes one of nine prediction modes involving linear combinations of the samples. For the INTRA-16x16 classes, four prediction modes are supported [1].

### 2.3 Performance and computing time complexity analysis

The complexity of the H.264/AVC encoder application depends on the algorithm, the encoding option tools, the input sequences and the architecture in which it is implemented. The encoding option tools are representative of the standard innovative features. When combining the standard new coding features, the implementation complexity accumulates, while the global compression efficiency becomes saturated [3]. To find an optimal balance between the coding efficiency and the implementation cost, a proper use of the H.264 new tools is needed to maintain the same coding performance as the most complex reference configuration (where all the coding options are on) while considerably reducing complexity.

To get the most efficient configuration, we performed in a previous work [16] a high level performance and complexity analysis of the major encoding tools on performance and computing time complexity. The experiments have been performed on a General-Purpose Processor (GPP) 1.6 GHZ INTEL Centrino platform using the JM 10.2 software reference version [17] with a main profile @ level 4. In comparison with the most complex configuration, one order of magnitude in complexity reduction has been achieved with less than 10% average bit rate increase for all the CIF and QCIF used test video sequences [16]. However and even for the very low bit rate QCIF "bridge far" sequence, the associated complexity in frames per second (fps) to compute the encoding algorithms on the GPP platform is of 2.16 fps. Even with this configuration offering an optimal trade-off between coding efficiency and implementation complexity, we are still very far from a real time performance of 25 frames per second. Implementing such an encoder represents a big challenge for resource-constrained multimedia systems since this requires very high computational power to achieve real-time encoding.

## 3. Parallelization of the H.264/AVC video encoder

To speedup the computing of this encoder, a multiprocessor implementation is probably needed. Several works have been elaborated for the parallelization of video coding standards to improve their execution performances and to achieve real time encoding or decoding. This section first reviews related parallelization works, and then presents our high-level independent target-architecture parallelization approach that is used to get an optimized parallel model of the H.264/AVC encoder with the best computation and communication workload balance.

### 3.1 Related works

Many parallel specifications of MPEG video encoders [4 and 5] have been performed for many distributed multi-core systems. For these specifications, a very coarse grain communication level, namely the Group Of Pictures GOP level, is used. This communication granularity has been justified by the little data dependency between processes acting on separate Group Of Pictures (GOP). This is due to the light data correlation from one GOP to another for the studied MPEG standards. For an embedded multiprocessor System on Chip (SoC), the GOP communication granularity is not appropriate given the limited on-chip memory resources. A more fine grain communication granularity should thus be selected for an embedded multiprocessor SoC implementation.

Typically, different types of data dependencies are built into the H.264/AVC video standard. These dependencies limit the concurrency space for a processes network using a task-pipelining parallelization method. In [6], it is proved that such a pipelining method is not enough appropriate particularly for the H.264/AVC since this standard presents very important data dependencies given the high spatial and temporal correlation between frames and macro-blocks of the same frame. For this, the use of data-level parallelism has been actually more

highlighted than the task-level pipelining. Using different data partitioning techniques, several parallel specifications of the H.264/AVC encoder have been proposed at a slice, MBs row, and MBs region granularity levels [7, 8, and 9]. Chen in [7] used a data partitioning technique to get parallel algorithms of a H.264/AVC encoder at slice granularity level based on Intel Hyper-Threading architecture. In this work, a frame is split into several slices, which are processed by multiple threads running on a system of 4 Intel Xeon processors with Hyper-Threading technique. Using this method, additional bit rate overheads have been generated. This is because of two main reasons: first, splitting frames into slices increases the bit information for the slice header, second motion estimation and compensation are not used for macro-blocks (MBs) between slices, which increases bits for the transformed coefficients.

To cope with these problems, other parallelization works have been oriented towards data splitting at the MBs row and MBs region granularity levels [8, 9]. For example, in [8] and based on the analysis of data dependency in the H.264 standard, input video data are mapped onto different processors at MBs row level. Using the Wave-front parallelization technique [8], each frame is first partitioned into MBs rows. Given that a MB can't be processed until its left neighbor in the same MB row is encoded, all MBs in the same MB row are then processed by the same process to reduce the data workload exchanges between processes. In addition, in [9] Sun selected to parallelize a H.264/AVC encoder using data parallelism at a MBs region level. Based on the analysis of different types of the encoder data dependencies, he proposed to split a frame into several MBs regions in which each region is composed by several adjoining columns of MBs. After that, these MBs regions are mapped onto different processes using the Wave-front technique. The data is exchanged appropriately between processes according to the analyzed data dependencies. These two last data-partitioning-based works [8, 9] showed that the computing performance and compression speed are better compared to the others already presented works and improved linearly with the number of the used processors. However, for these parallel specifications, the decomposition of tasks depends of the number of the simulated processors and the computation workload is observed after software simulation on a multiprocessor platform.

### 3.2 The proposed parallelization approach

For a H.264/AVC encoder, it is observed that the intra prediction and inter prediction modules are two independent components which can operate concurrently. For a more efficient parallelization, a task-level parallel execution of the most time-consuming computational components should be exploited. The goal of this step is to extract the available task-parallelism from the application by splitting compute nodes as far as possible to get a valid parallel model. For an optimal design flow, our aim is to provide a parallel specification which forms a good starting point for mapping onto different systems-on-chip platforms. To do so, we proposed in a previous work a high-level independent target-architecture parallelization approach [11] to get an optimized parallel model of the encoder suitable for embedded SoC implementation. The key characteristics of the proposed approach is the use of the parallel streaming programming models of computation, the selection of a fine-grain communication at a Macro-Block granularity level, and the exploration of the data and task levels forms of parallelism simultaneously,.

Among the existent streaming models of computation (MoC), our approach is based on the use of the Kahn Process Network (KPN) [18] model implemented by the Y-chart Applications Programmers Interface (YAPI) C++ run time library [19]. The KPN MoC assumes a network of concurrent autonomous processes that communicate in a point-to-point fashion over unbounded first-in-first-out (FIFO) channels. Read actions from these FIFOs block until at least one data item becomes available. It is demonstrated that the execution of a Kahn Process Network is deterministic and independent of process interleaving, meaning that for a given input always the same output is produced and the same workload is generated, irrespective of the execution schedule. For this reason, an application programmer can combine processes that represent signal processing functions into process networks without specifying their order of execution. Moreover, a system designer can exploit the concurrency between the processes by using processing elements that operate in parallel. Finally, the key characteristic of the KPN model is that it specifies an application in terms of distributed control and distributed memory which allows mapping the application onto a multiprocessor platform in a systematic and efficient way.

The second considered characteristic of our proposed approach is the granularity at which data is communicated. The optimal communication granularity between tasks needs to be correct evaluated in order to prevent tasks from long waiting and to avoid spending too much time on synchronization. As presented in section 3.1, many previous task-level parallelization works have been performed for different encoding applications. For these studies, The GOP, slice or frame level communication granularity has been used. It has been shown that for embedded System-on-Chip implementation, neither the GOP or frame level granularities are viable. For such systems, the best granularity may be the fine grain level, i.e. at the Macro block (MB) level. The use of such a granularity requires only the current and reference frames to be stored. Each frame is considered as the current workload, and the encoding process of each frame is divided between the processors.

The availability of many processing cores speeds-up the system execution by introducing several types of parallelism. The predominant forms of parallelism are Data Level Parallelism (DLP) and Task Level Parallelism (TLP). In our case, these two forms of parallelism are used to get an efficient parallel implementation of the encoder. The DLP has perhaps been the most commonly used form of parallelism. This consists in extracting from the original source code regular data (vectors, matrices, etc.) on which the same sequence of instructions are applied. The TLP consists in partitioning the code by functionality. Optimized task-level decomposition shall regroup some system functionalities to get a balanced task computation workload. The effective implementation of a TLP parallelization requires a data dependency analysis between tasks and an accurate study of their execution orders from the original sequential code. For our case, simultaneous exploration of the two predominant DLP and TLP forms of parallelism are used. This means that communication and computation workload analysis are used to provide a global guidance when optimizing concurrency between processes. In general, when the concurrency bottlenecks are identified, task and data levels splitting and/or merging are performed for better distributing the computing workload over the processes. For the most computational-expensive tasks, data splitting is proposed for a better concurrency optimization [11].

## 4. The starting parallel specification of a H.264/AVC video encoder

The proposed approach discussed above has been used to derive in a structured way an optimized parallel specification of the H.264/AVC encoder. This has been performed according to two steps. In the first step, a starting parallel KPN model is obtained by task-level decomposition. The second step considers concurrency optimizations of the starting model using task-merging and data-partitioning forms of parallelism to derive a model with the best computation and communication workload balance. This section will discuss some of the aspects and issues used for the development starting model.

### 4.1 The first starting KPN/YAPI parallel model

The Task Level Parallelism (TLP) is first considered. The goal of this step is to extract the available task-parallelism by splitting compute nodes as far as possible to get a starting valid parallel KPN model of the encoder. For this case, the block diagram of the figure1 has served as a starting point for extracting the task-level parallelism. The proposed model is given figure 2.

The "VidIn" process shown in figure2 represents the input of the encoder. This process is responsible for collecting the video data (YUV frames) from the input file (video sequence with YUV format), the frame width and height dimensions, the total frame number, and the frame rate information. Each frame is divided into "YUVMB" MBs of 16x16 pixels. The "Dmx" process forwards these macro-blocks to the "Sub", "Mec", and "Intra-Pred" processes. The "Sub" process reads the predicted "PredYUVMbToSub" MB, subtracts it from the current "YUVMbToSub", and sends the residual data "YUVMbToDCT" to the "Dct_Dec" process to perform associated transforms, respectively on the Y luminance and the UV chrominance MBs. These MBs are first arranged into blocks of 4x4 pixels. Each 4x4 block is first transformed into DCT coefficients using an appropriate integer transform then Q quantized and sent as "QuanMb" to the "Vlc" process. The "Dct_Dec" is also responsible for decoding "QuanMb" via a rescaling and an inverse transform and transmitting the "DecMb" to the "Add" process. The "Vlc" receives the quantized DCT coefficients "QuanMb", performs the CAVLC entropy coding and transmits the resulting "BitStreamFrm" compressed bit stream to the "VidOut" that sends the H.264 compressed data bit stream to the output file (.h264).

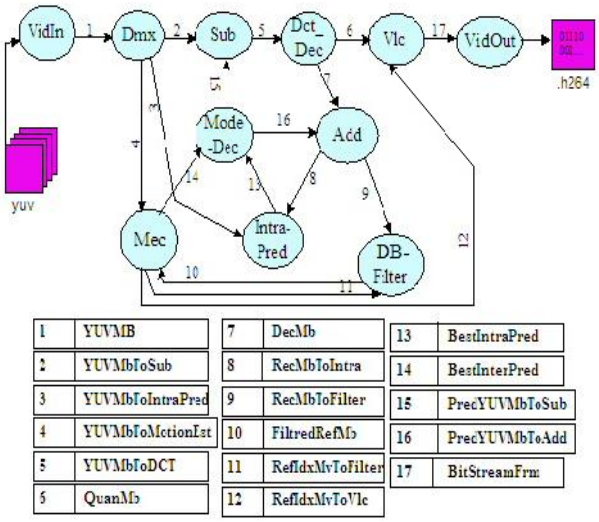


**Figure2.** Starting parallel H.264/AVC KPN model

The "Add" process uses the residual decoded "DecMb" MBs and the best inter or intra predicted MBs "PredYUVMbToAdd" to reconstruct the previously encoded "RecMbToIntra" (but un-filtered) MB. Using the current MB "YUVMbToIntraPred" and the reconstructed previously encoded MB "RecMbToIntra", the "Intra_Pred" first maintains the "RecMbToIntra" MB in the reconstructed frame, then, performs an intra-prediction on each MB using 9 prediction modes for the 4x4 luma blocks, 4 prediction modes for the16x16 luma blocks, and 4 modes for the 8x8 chroma blocks. The best intra-prediction mode cost obtained and the associated predicted MB "BestIntraPred" are sent to the "Mode_Dec" process.

Parallel to the intra-prediction process, each "Dmx" output "YUVMbToMotionEst" current MB is inter-

predicted using one or more reference frames by the "Mec" process. This process is also responsible for maintaining the reference frames memory. The list of past frames is generated through the filtered reference MBs received as an output of "DB_Filter" process. The "DB_Filter" receives the reconstructed decoded "RecMbToFilter" MBs (only these used as reference) from the "Add" process and information about the references indexes and the motion vectors of this MB (already inter-predicted) from the "Mec" process. Then, filtering is applied on each reconstructed previously encoded MB (before storing the macro-block for future predictions) to reduce blocking distortions. The best inter-prediction mode cost obtained along with the corresponding predicted MB are sent as a "BestInterPred" structure to the "Mode_Dec" process. There are also information about the frame motion vectors and its reference indexes which are copied to the "Vlc" process via the "RefIdxMvToVlc" channel. Using the best intra-prediction and inter-prediction modes, the "Mode_Dec" process selects the best optimal "PredYUVMbToSub" predicted MB of them and transmits it to the "Sub" process.

## 4.2 The KPN/YAPI programming strategy

For the implementation of the parallel model of figure2, we started with the sequential C reference code of the fixed configuration defined in section 4. The sequential code is modified and structured by hand to describe the KPN in C++. Each Kahn process is described by a set of associated functions extracted from the original C code. The inter process communication is performed using solely the YAPI I/O FIFO primitives. Using global variables is not allowed with a KPN model [18]. Thus, to ensure inter process communication, all the global shared variables used in the sequential reference code are grouped into associated data structures for communication over FIFO channels.

For efficient task-level decomposition, a YAPI programming strategy is proposed. This strategy is based on the analysis of the role played by each process in the proposed KPN model. Given this, all the process related functions and data structures are extracted from the sequential source code. The steps of the used strategy are exactly as follows. (1) Definition of the code of each process including all related functions, local and global data structures. (2) Once the extracted code is compiled with no errors, we precede to its reimplementation using solely the YAPI C++ syntax. Global variables are converted into local variables transmitted over FIFO communication channels. (3) Finally, the behavior validation is performed. This step consists in checking for each process of the model that the associated separate code carries out the same computation with the same functionalities as the old sequential source reference code.

## 4.3 Important YAPI implementations issues

For the effective YAPI implementation of the starting KPN model, the following issues are considered to deal with the reference frame memory management; the large inter process data structures, the specialized function redundancy and the management of the large local variables.

### *4.3.1 Reference frame memory management*

For the model of figure2, there are some processes exchanging Macro Block (MB) data streams, but there are others like the motion estimation compensation "Mec" process that is accessing data from reference frames. For a "Mec" process, a full reference frame has to be transmitted over a dedicated FIFO Channel. First, transmitting full reference frames from the "DB_Filter" process to the "Mec" process is considered. In this case, for low resolution QCIF video format, about 38 Kilo bytes of data are needed for each reference frame. For the used encoding configuration, the reference frame number is fixed to 3. This results in a minimum of 115 Kilo bytes of needed FIFO size memory for transmitting the reference frames between "Mec" and "DB_Filter" processes. Typically, this is not practical, particularly for higher resolution video frames. Given this and as using global shared variables is not allowed, we opted for the "Mec" process to handle locally the reference frames and to maintain the memory of past frames. After having received all the filtered intra MBs, the "Mec" starts the inter prediction of the P/B-type MBs one by one starting from the left. In this case, a problem of processing a P/B-type MB would persist: the "Mec" process needs the "FilteredRefMb" information from the "DB_Filter" that can not be processed before having received the "RefIdxMvToFilter" data from the "Mec" process, as shown in our KPN model of figure2.

To resolve these dependencies of reading the filtered MBs "FilteredRefMb", we first proposed to proceed as follows. Once one P/B-type $MB_i$ is read, the "Mec" process remains blocked until the filtered reference "FilteredRefMb" $MB_{i-1}$ of the previous inter-prediction is read. Then, the "Mec" process stores the obtained filtered reference $MB_{i-1}$ in the associated reference picture, performs the motion estimation and compensation of the current $MB_i$, and then transmits the motion vectors data "RefIdxMvToFilter" to the "DB_Filter" to filter these previously inter-predicted $MB_i$. Such a solution increases the overhead and the data dependencies between the pipelined tasks, and considerably decreases the effective parallelism gain. To cope with this, we proposed to delay reading the filtered MBs of currently processed "Mec" frame. For this case, just before starting the inter prediction of the first MB of the next frame, the "Mec" process starts reading the filtered MBs of the previous frame, then reorders the list of reference frames DPB (Decoded Picture Buffer). However, such a solution is not convenient

because the FIFO channel connecting the “Mec” to the “DB_Filter” processes has to support for a minimum of one frame size. Finally, to resolve the entire problem, we modified the implementation of th “Mec” process as follows. After inter-predicting one row of MBs beginning from the left and before processing the first MB of the next row, the “Mec” process reads the filtered MBs of the previously encoded row, and stores them in an associated local reference picture. Once a reference frame is entirely read, it will be added in the DPB list of past frames.

#### *4.3.2 The inter process communication data structures*

For each inter process communication, we associated for a needed data a sort of token data structure that will be transmitted over a dedicated FIFO channel. As previously discussed, the largest structure used is the Macro-Block data structure. Such a data structure is transmitted between the “VidIn” & “Dmx”, “Dmx” & ”Sub”, “Sub” & ”Dct_Dec”, “Dct_Dec” & ”Vlc”, “Dct_Dec” & ”Add”, “Add” & ”DB_Filter”, “Dmx” & ”Mec”, and “Dmx” & ”Intra-Pred” processes. This structure is formed essentially by the Y luminance and the UV chrominance data blocks, the MB horizontal/vertical positions, the type of MB (I, P, or B), AC & DC coefficients, “is or is not reference” status, the neighboring MBs positions and addresses, and other information used by the “Mec” and “Intra-Pred” processes. In addition to data transmitted between processes via dedicated communication channels, some constant parameters are also needed. For example, the Sequence Parameter Set (SPS) is a constant parameter used for the whole sequence and the Picture Parameter Set (PPS) is a constant needed for each frame. Exchanging such constant parameters over dedicated FIFOs or adding them to associated token data structures will add significant overhead to the system. For this case and given the C++ concepts that characterize the YAPI run-time environment, we opted for defining such constant parameters as private members in the YAPI class of the whole Process Network. These private members will be used by any Process Class of the model without needing to be exchanged over associated channels.

#### *4.3.3 Specialized functions Redundancy*

Using the original C source code of the JM10.2 reference version [17], there are particular specialized functions used by multiple processes. For example, the “GetNeighbour()” function is used to gather positions information on the neighboring Luminance and Chrominance blocks. This function and all its children are needed by all the “Intra-Pred”, the “Mec”, and the “Vlc” processes. For the use of these specialized functions, the first option consists in their implementation into a dedicated specialized process along with a dedicated FIFO to each associated process. However, implementing this option leads to a maximum communication overhead and an important data dependency between processes. To minimize this overhead, we opted for a redundant implementation of all the specialized functions at the cost of more computing burden of the associated processes.

#### *4.3.4 Large local variables management*

Using the YAPI development framework, a separate private stack space is allocated for each process of the network. This stack is used to store the intermediate results, the local variables, and all functions call management. For the case of the H.264/AVC encoder, there are several large local video data structures that are needed to be allocated on the stack. As the total amount of stack space of each process is fixed to 64 Kilo bytes, this may be insufficient and may result in a “stack overflow” [20]. Such a stack overflow will lead to an access violation that causes the program to be killed and a core dump to be generated [20]. To cope with this, all the stack large data structures have been allocated dynamically on the heap using the “malloc” and “new” dynamic allocation services.

### 4.4 Performance evaluation of the starting KPN/YAPI model

The parallel KPN model of figure2 is implemented using the YAPI multi-threading programming environment. The implemented model is first validated by high level functional simulation. The correctness of the parallelized code is proved by comparing both execution results of sequential and parallelized code using the same test benches. For a QCIF “Bridge close” sequence of 13 YUV frames, communication and computation workload analysis has also been considered to identify the potential bottlenecks and thus to provide a global guidance when optimizing concurrency between processes. The obtained communication workload results are given the next figure 3

| | size | Tsize | Wtokens | Wcalls | T/W | Rtokens | Rcalls | T/R |
|---|---|---|---|---|---|---|---|---|
| h264.YUVMB | 1182 | 40752 | 1287 | 1287 | 1 | 1287 | 1287 | 1 |
| h264.YUVMbToSub | 2 | 40752 | 1287 | 1287 | 1 | 1287 | 1287 | 1 |
| h264.PredYUVMbToSub | 1 | 640 | 1287 | 1287 | 1 | 1287 | 1287 | 1 |
| h264.PredYUVMbToAdd | 1 | 640 | 1287 | 1287 | 1 | 1287 | 1287 | 1 |
| h264.YUVMbToDCT | 1 | 40752 | 1287 | 1287 | 1 | 1287 | 1287 | 1 |
| h264.QuanMb | 1 | 40752 | 1287 | 1287 | 1 | 1287 | 1287 | 1 |
| h264.DecMb | 1 | 40752 | 1287 | 1287 | 1 | 1287 | 1287 | 1 |
| h264.RecMbToIntra | 1 | 592 | 1287 | 1287 | 1 | 1286 | 1286 | 1 |
| h264.YUVMbToIntraPred | 1 | 40752 | 1287 | 1287 | 1 | 1287 | 1287 | 1 |
| h264.BestIntraPred | 1 | 648 | 1287 | 1287 | 1 | 1287 | 1287 | 1 |
| h264.RecMbToFilter | 8 | 40752 | 693 | 693 | 1 | 693 | 693 | 1 |
| h264.RefIdxMvToFilter | 9 | 416 | 594 | 594 | 1 | 594 | 594 | 1 |
| h264.RefIdxMvToVlc | 1 | 304 | 1188 | 1188 | 1 | 1188 | 1188 | 1 |
| h264.FiltredRefMb | 1 | 1032 | 693 | 693 | 1 | 693 | 693 | 1 |
| h264.YUVMbToMotionEst | 1 | 40752 | 1188 | 1188 | 1 | 1188 | 1188 | 1 |
| h264.BestInterPred | 1 | 648 | 1188 | 1188 | 1 | 1188 | 1188 | 1 |
| h264.BitStreamFrm | 12 | 40 | 13 | 13 | 1 | 13 | 13 | 1 |

**Figure3.** Communication workload of the starting parallel model

This figure describes the total number of Write tokens (Wtokens) and Read tokens (Rtokens) exchanged over all the used data channels of the network. The “Tsize” for one token represents the average amount of data communicated per call between two processes. For the QCIF “Bridge close” 13 frames sequence, each frame consists of 99 MBs

of 16x16 pixels. One MB is representative of Luminance (Y) and Chrominance (UV) data in a 4:2:0 format. One MB is constituted with two 8x8 blocks of chrominance, and one 16x16 block of luminance. For the implemented YAPI model, we have 1287 (99*13frames) intra and inter MBs communicated over the "YUVMB" FIFO channel from the "VidIn" process to the "Dmx" process. Given the "Tsize" of one token (40752 of 1 byte size), the total bytes number communicated over this "YUVMB" channel is 1287*40752*1 bytes. For the "YUVMbToMotionEst" channel, there are only 1188 inter predicted and bidirectional MBs sent from the "Dmx" process into the "Mec" process. Given the results of figure3, it is clear that the communication workload is somewhat unbalanced for this starting computational network. The very large exchanged data structures are outputs of the "Dmx", "Sub", "Dct_Dec", and "Add" processes. The remaining tokens exchanged between the others tasks are all balanced.

A computational workload analysis has been also considered using the "Gprof" GNU [21] profiling tool. The obtained results are reported in figure 4 in terms of the CPU time percentage spent in the process execution. Given the profiling results of figure4, it is clear that the computational workload of the model is too much unbalanced. Some processes have negligible complexity; others especially the "Mec" is very complex. Although we fixed the optimal parameters configuration using a fast full search algorithm, a search range of 8, 4 variables block sizes, and only 3 reference frames [16], the "Mec" process of the model is still a very computational-expensive with more than 50% of the total computing time complexity.

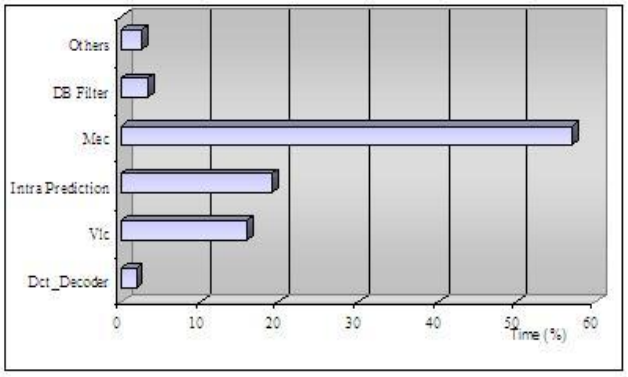


**Figure4.** Parallel computational profiling of the first proposed model

Finally it is clear, using the obtained communication and computation workload results, that the starting model of figure2 does not have good concurrency properties. This outlines the potential of using different steps of task level splitting or merging and data level partitioning to derive in a structured way a parallel implementation of the H.264/AVC encoder that has a balanced computational workload and good communication behavior.

# 5. Concurrency optimization of the starting H.264/AVC parallel model

To get a parallel implementation of the encoder that has a balanced computational workload and good communication behavior, different steps of the task level merging and data level splitting have been performed. The task-merging is used to merge the "Dct_Dec", the "DB_Filter", the "Sub", the "Dmx", and the "Add" processes into only one "Dct_Dec_Filter" process. In this case, the associate channels transmitting very large token structures are removed. For the most computational-expensive "Mec" task, data splitting is proposed for better concurrency optimization. For the most computational-expensive Motion estimation and compensation "Mec" task, a data partitioning strategy has been considered to distribute the computing of this process into three "Mec1", "Mec2", and "Mec3". In addition, tripling the "Mec" processes will result in tripling the associated Input/Output FIFO channels. For partitioning the data to the three "$Mec_i$" modules, a strong data dependency analysis of the motion estimation and compensation process has been performed. Given the results of this analysis, an appropriate strategy has been used for the best partitioning of the communicated data to the motion estimation compensation (MEC) processes and for maximizing the parallelism rate between the triple decomposed MEC processes. Further details about the performed data dependency analysis along with deep discussions of the main problems encountered and the proposed solutions for the effective data-splitting of the three inter-prediction modules are discussed in our previous work presented in [11].

Finally, the obtained optimized parallel model is given in figure 5. This figure clearly shows the task-merging of the "Dct_Dec_Filter" process, and also the data-partitioning for the "Mec" process into the "Mec1", "Mec2", and "Mec3" processes with the appropriate connections between the "$Mec_i$" processes and their environment. This model has been implemented and validated at YAPI system level. The communication and computation workload profiling results of the final parallel model for the same QCIF "Bridge close" 13 YUV frames sequence, are reported in figure 6.

It is clear given the obtained communication profile that the optimized model has better communication behavior compared to the starting model. In addition, as shown in figure 6, the data partitioning of the MEC processes comes with a decrease in the computational burden of these processes, and thus a better computational workload balance of the model is observed. The final proposed model has obviously better communication and computational behavior compared to the first starting model. Anyway, one can further use the data parallelism for the MEC

module to more reduce the computational workload on the associated processes.

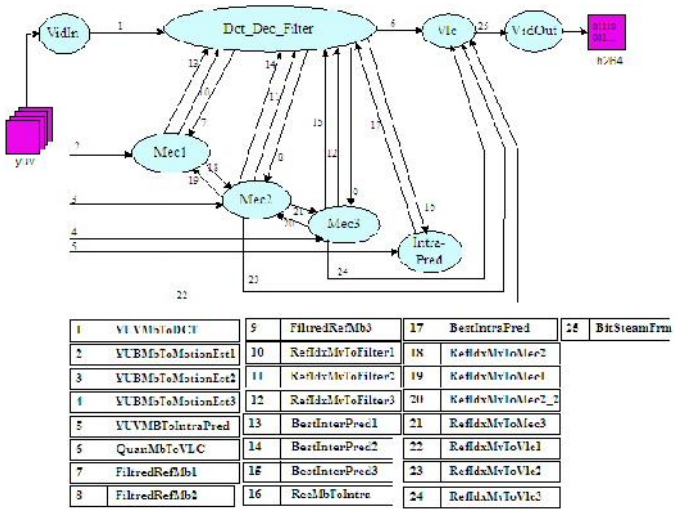


**Figure5.** Proposed optimized parallel KPN model of the H.264 encoder

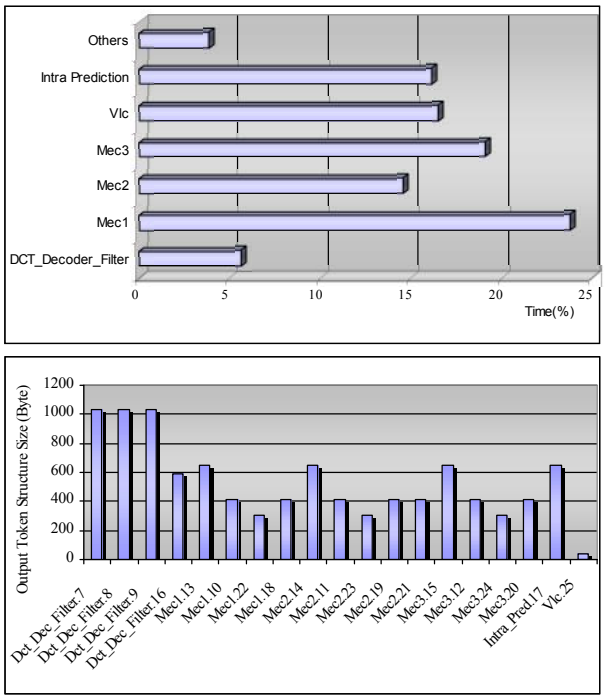


**Figure6.** Computation and Communication workload profiling of the optimized final parallel model

# 6. Multiprocessor simulation results

To demonstrate the effectiveness of the proposed optimized parallel model of the H.264/AVC encoder, the system-level simulation and modeling framework Sesame/Artemis [22] has been used to evaluate the encoding performance, in terms of number of coded frames per second, targeting several multiprocessor platforms. For this, the base target architecture and the mapping strategy are first presented. Sesame system level design is then used for performance evaluation of the encoder targeting multiple multiprocessors architectures.

## 6.1 The base target architecture and mapping

Using the Sesame system-level design methodology, three software model specifications are required: the application process network model, the target architecture model, and the mapping model of the application onto the architecture. For this, the optimized parallel model given in section 5 has been first ported into the Sesame framework by transforming the YAPI model into a C++ PNRunner (Process Network Runner) application. The network model is then simulated with the PNRunner simulator to generate computational and communication event traces of the application execution. These event traces are called trace-event queues. For further details about the Sesame/Artemis simulation and modeling framework, one can refer to [22] and [23].

Parallel to the application model specification, the target architecture is modeled with the Pearl object-based simulation language [24]. For this, the Sesame/Artemis framework provides a small library of architecture component models. These models consist in black-box base models of processing cores, generic buses, generic memories, and several interfaces for connecting these base model building blocks. Once a target architecture model is validated, a trace-driven simulation of the application events traces queues mapped to the architectural components is carried out. Such a simulation requires an explicit mapping of the KPN processes and channels to the particular components of the target architecture. More than one KPN process can be mapped to a same processor. In this case, the system simulator automatically schedules the events from the different queues [25].

In our case, the base target architecture is given in figure 7. This figure represents a multiprocessor platform communicating with a shared DRAM memory through a common bus. For this platform, we have used general purpose processors (assumed to be MIPS R3000), and assumed a conservative timing of 100ns to read/write a 64-bit word from/to DRAM. The instruction latencies for the MIPS R3000 microprocessors components were estimated using technical documentation. Communication between components is performed through buffers in shared memory. For sufficient design space exploration, several platform models are used. The platforms differ by the number of used processors. One platform is used with two processors; a second is with four and a third is tested with six processors.

Mapping application processes to this platform has been decided explicitly given the obtained computation and communication load distribution results of figure6. For the bi-processor platform example, the total computation load has been distributed between the two processors. The “Mec1”, “Mec2”, and “Dct_Dec_Filter” processes are mapped to one processor, and all the others to the second

processor. Using the four-processor platform, the used mapping strategy is showed in figure 7. In this case, first, the most complex "Mec1", "Mec2", "Mec3", and "Intra-Pred" processes are mapped separately to each used core to guarantee a competitive execution between them. Then, the "Dct_Dec_Filter" process is added to run with the "Mec2" process on the same core. The "Vlc" is also added to the "Intra-Pred" process and is mapped to the fourth processor.

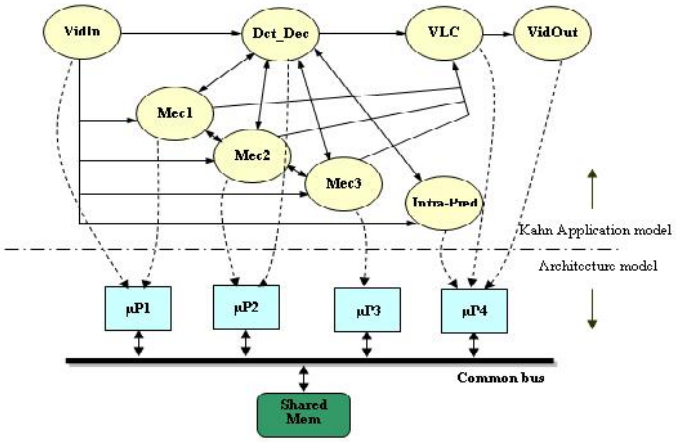


**Figure7.** H.264 encoder's application to architecture mapping

## 6.2 Performances evaluation of the proposed parallel model

The PNRunner optimized H.264/AVC network model of figure 5 is mapped into the different used platforms, then a performance analysis is performed by system-level simulations. In all the experiments, the input test video sequence consists of YUV frames with a QCIF resolution of 176*144 pixels. The simulation results of the QCIF "Bridge-close" sequence H.264/AVC encoding process are obtained for different platforms and are presented in the next figure 8. It is clear from this figure, that the encoding performance obtained, in terms of frames per second, are getting linearly better with the number of simulated microprocessors. For each case, as the application model is considered to be optimal, the execution/communication performances gain may be improved by changing the mapping policy or the platform architecture. To modify the architecture, a designer can explore the use of other communication models or enhance the architecture with hardware components using appropriate HW/SW partitioning.

In addition, the encoding performance results of the optimized model have also been compared to those previously obtained using the data-level parallelization approaches proposed in [8 and 9]. Results of this comparison are given in the table1. This table clearly shows that our solution, based on simultaneous task and data level parallelism, has achieved better performance of the encoding process. Actually, using references [8] and [9], data splitting is performed respectively at the MBs row and MBs region communication granularity levels. But for our case a more fine-grain Macro-Block communication granularity level is exploited. Thus, with a more fine grain data amount exchanged by the processors, our proposed approach is more adapted for embedded multiprocessor SoC implementations having limited on-chip memory resources.

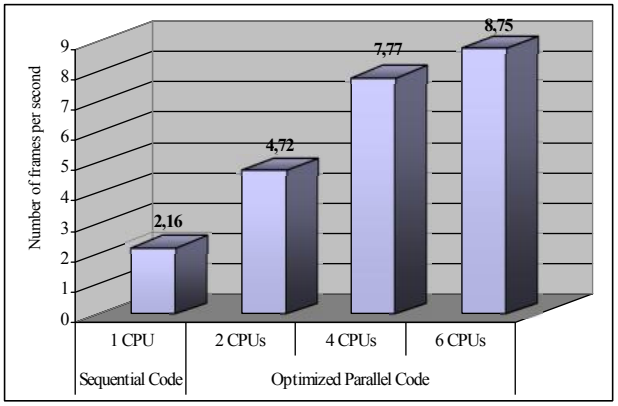


**Figure8.** Sesame/Artemis H.264 encoding performances vs. number of simulated processors

**Table1.** Obtained Multiprocessor simulation results in comparison to those obtained in [8] and [9]

| | *Number of processors* | *Encoding simulation time of the 7 QCIF YUV frames (s)* | *Number of frames per second (fps)* | *Speed up* | *Speed up in [8]* | *Speed up in [9]* |
|---|---|---|---|---|---|---|
| *Sequential H.264 code (JM10.2)* | Mono-Processor | — | 2.16 | 1 | 1 | 1 |
| *Optimized parallel H.264 model* | 2 Processors | 1,6 | 4.72 | 2.19 | — | — |
| | 4 Processors | 1.00 | 7.77 | **3.6** | 3.1 | 3.3 |

Finally, for the four-processor platform with the common bus structure, performance numbers for the execution/communication workload is obtained for each used architecture component. The obtained results are shown in figure9. For each component, a bar shows the breakdown of the time spent on reading/writing, being busy and being idle.

Using figure 9, it is obvious that the computation cost is much more important than the time spent in reading/writing from/to the shared memory. The communication and computation loads are nearly balanced for all the used components. Such a result confirms the good concurrency properties of the proposed optimized parallel model and the good mapping policy used. However, the times being idle are too much important in comparison with those being busy for all the architecture components. This has probably caused a substantial degradation of the final encoding performances. Given the important amount of data communicated between processes for this encoding process, it is clear that the common memory bus structure constitutes a serious communication bottleneck. Actually, the very important data dependency between processors requires a potential memory access and allocation for the read/write operations. For a common-bus

multiprocessor architecture, this causes a saturation of bus and thus a lot of time is spent in waiting to read/write data from/to other component. For further design space exploration and in order to reduce the communication bottleneck observed for the common-bus-based architecture, others inter processors communication structures and topologies need to be evaluated for a better encoding performance.

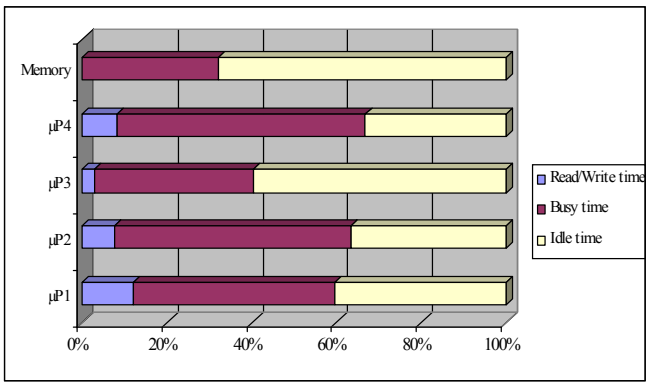


**Figure9.** Reading-Writing/Execution/Idle statistics for the common-bus-based architecture

## 7. Conclusions

The H.264/AVC has been designed with the goal of enabling significantly improved compression performance relative to all existing video coding standards. Implementing a H.264/AVC video encoder represents a big challenge for resource-constrained embedded systems since this requires very high computational power to achieve real-time encoding. For a cost-effective implementation, a multiprocessor approach is necessary to share the encoding execution time between several processors. Prior to the multiprocessor implementation, the sequential H.264/AVC reference code has to be distributed using appropriate high level parallel programming model of computation. For this purpose, a high-level independent target-architecture parallelization approach is proposed. The key characteristics of this approach is the use of the parallel streaming programming models of computation, the selection of a fine-grain communication at a Macro-Block granularity level, and the exploration of the data and task levels forms of parallelism simultaneously.

Using the reference C code, a staring parallel model of the encoder is proposed. This model, based on Kahn Process Network (KPN) model of computation, is implemented using the YAPI multi-threading programming environment, and its associated functional simulation results are obtained. Given the high complexity of the H.264/AVC standard, details about implementation issues are given to cope with problems related to reference frame memory management; large inter process data structures, specialized function redundancy and management of the large local variables. According to the communication and computation concurrency properties of the implemented starting model, task splitting or merging and data level parallelism have simultaneously been explored to derive in a structured way an optimized parallel YAPI/KPN model with a good computation and communication workload balance.

To evaluate the effectiveness of the optimized parallel model, the system-level Sesame/Artemis simulation framework has been used targeting multiple multiprocessors platforms. It has been shown that the encoding performance obtained, in terms of frames per second, are getting linearly better with the number of simulated processors. In addition and in comparison to previous parallelization approaches proposed in [8 and 9], the proposed optimal model achieves better performance and is more appropriate for use in SoC implementation since it is based on a more fine grain communication granularity level. Finally it has been shown, for the four-processor platform with the common bus structure, that the computation cost is much more important than the time spent in reading/writing from/to the shared memory. The communication and computation loads are nearly balanced for all the used components. These results represent again a solid confirm of the good concurrency properties of our optimized model.

## References

[1] Iain E.G. Richardson, H.264 and MPEG-4 Video Compression: Video Coding for Next-generation Multimedia, John Wiley &Sons Ltd, 2003.

[2] M. Alvarez, A. Salami, A. Ramirez, M. Valero, A Performance Characterization of high Definition Digital Video Decoding using H.264/AVC, *in: Proc. 2005, IEEE International, Symposium on Workload Characterization*, pp. 24 – 33.

[3] S. Saponara, K. Denolf, G. Lafruit, C. Blanch, J. Bormans, Performance and Complexity Co-evaluation of the Advanced Video Coding Standard for Cost-Effective multimedia communication, *EURAPIS Journal on Applied Signal Processing*, pp. 220-235, 2004:2.

[4] K. Shen, L.A. Rowe, and E.J. Delp, a Parallel Implementation of an MPEG1 Encoder: Faster than Real-Time, *in: Proc. 1995, SPIE Conference on Digital Video Compression: Algorithms and Technologies*. San Jose.

[5] S. Bozoki, S.J.P. Westen, R.L. Lagendijk, and J. Biemond, Parallel algorithms for MPEG video compression with PVM, *in: Proc. 1996, International EUROSIM Conference*: Delft. The Netherlands 315-326.

[6] A. Gulati, G. Campbell, Efficient mapping of the H.264 encoding algorithm onto multiprocessor DSPs. In*: Proc. of SPIE-IS&T Electronic Imaging* (2005)

[7] Y-K. Chen, X. Tian, S. Ge, and M. Girkar, Towards Efficient Multi-Level Threading of H.264 Encoder on Intel Hyper-Threading Architectures, *In proceeding: 18th International Parallel and Distributed Processing Symposium* (2004).

[8] Z. Zhao, P. Liang, A Highly Efficient Parallel Algorithm for H.264 Video Encoder, in proceeding*: 31st IEEE International Conference on Acoustics, Speech, and Signal Processing* (2006).

[9] Sh. Sun, D. Wang, and S. Chen, A Highly Efficient Parallel Algorithm for H.264 Encoder Based on Macro-Block Region Partition*, HPCC 2007*, LNCS 4782, pp. 577–585, Berlin Heidelberg 2007.

[10] H. Krichene Zrida, A. C. Ammari, A. Jemai, M. Abid. "A YAPI-KPN Parallel Model of a H.264/AVC Video Encoder". *4th IEEE International Conference on Ph.D. Research in Microelectronics and Electronics (PRIME)*, 22-25 Juin 2008, Bogaziçi University, Istanbul-Turkey.

[11] H. Krichene Zrida, A. Jemai, A.C. Ammari, M. Abid, "High Level H.264/AVC Video Encoder Parallelization for Multiprocessor Implementation", *12th ACM/IEEE Design Automation and Test in Europe conference and exhibition (DATE)*, Nice-France, 20-24 April 2009, IEEE Computer Society

[12] R. Schäfer, T. Wiegand, H. Schwarz, The emerging H.264/AVC standard, *EBU technical review* (January 2003).

[13] J. Ostermann, J. Bormans, P. List, D. Marpe, M. Narroschke, F. Pereira, T. Stockhammer, and T. Wedi. Video coding with H.264/AVC: Tools, Performance, and Complexity, *in: Proc. 2004, IEEE Circuits and Systems Magazine*.

[14] ISO/IEC 14496–10:2003, Coding of Audiovisual Objects—Part 10: Advanced Video Coding, 2003, also ITU-T Recommendation H.264 Advanced video coding for generic audiovisual services

[15] H. Malvar, A. Hallapuro, M. Karczewicz, and L. Kerofsky, Low-Complexity transform and quantization in H.264/AVC, *in: Proc. 2003, IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, pp. 598–603.

[16] H. Krichene Zrida, A.C. Ammari, A. Jemai, M. Abid, Performance/Complexity Analysis of a H.264 Video Encoder, *International Review on Computers and Software (IRECOS'07)*, Vol 2 n°4, pp n°401-414, July 2007.

[17] H.264 Reference Software Version JM 10.2, http://iphome.hhi.de/suehring/tml/. (November 2005).

[18] G. Kahn, The semantics of a simple language for parallel programming, *in: Proc. 1974, the IFIP Congress 74*, North-Holland Publishing Co.

[19] E.A. Kock, G. Essink, W.J.M. Smits, P. van der Wolf, J.-Y. Brunel, W.M. Kruijtzer, P. Lieverse, and K.A. Vissers, YAPI: Application modeling for signal processing system, *in: Proc. 2000, 37th Design Automation Conference (DAC'2000)*, Los Angeles, CA, pp. 402–405

[20] T. Stefanov, P. Lieverse, E.F. Deprettere, and P. van der Wolf. Y-chart based system level performance analysis: an M-JPEG case study. *Proceedings of the Progress workshop*, 2000.

[21] Susan L. Graham, Peter B. Kessler; and Marshall K. McKusick, Gprof: A Call Graph Execution Profiler, *in: Proc. 1982 the SIGPLAN '82 Symposium on Compiler Construction*. http://www.gnu.org/software/binutils/manual/gprof-2.9.1/

[22] J.E. Coffland and A.D. Pimentel, "A Software Framework for Efficient System level Performance Evaluation of Embedded Systems", *Proceeding: SAC "the ACM Symposium on Applied Computing",* Mar. 2003, Melbourne, Florida, USA.

[23] C. Erbas, "System-Level Modeling and Design Space Exploration for Multiprocessor Embedded System-on-Chip Architectures", PhD thesis at the University of Amsterdam, 2006

[24] A. D. Pimentel, P. Lieverse, P. van der Wolf, L. O. Hertzberger, et E. F. Deprettere, Exploring embedded-systems architectures with Artemis, IEEE Computer, vol. 34, no. 11, pp. 57–63, Nov. 2001.

[25] A.D. Pimentel, S. Polstra, F. Terpstra, A.W. van Halderen, J.E. Coffland, and L.O. Hertzberger , Towards Efficient Design Space Exploration of Heterogeneous Embedded Media Systems, 2001.

**Hajer KRICHENE ZRIDA** was born in Sfax, Tunisia, on January 03, 1979. She received an engineering degree in Computer Science in July 2003 from the National School of Computer Sciences (ENSI), Tunisia, and a MASTER degree in New Technologies of Dedicated Computer Systems from the National School of Engineers of Sfax (ENIS) in March 2005. Now, she is preparing a Ph.D. at the Micro technology and System on Chip laboratory in the National School of Engineers of Sfax, Tunisia. Her main areas of interest are the system-

level modeling languages, video encoding techniques and parallelization approaches for video coding applications in embedded multiprocessor systems.

**Ahmed Chiheb AMMARI** was born in Mahdia, Tunisia, on November 20, 1968. He received an engineering degree from the National School of Engineering in Monastir (ENIM), Tunisia, in 1993, and a Ph.D. degree in electrical engineering from the National Polytechnic Institute of Grenoble (INPG) in France in 1996. Since 1997, he has been an assistant professor of electrical engineering at the Institut National des Sciences Appliquées et de Technologies (INSAT) in Tunis, Tunisia. In parallel with his teaching duties, he has had the opportunity to work on many research projects dealing with system level design and performance evaluation for Multiprocessor Systems on Chip (MPSoC), human power harvesting for wearable computers, power aware mobile computing, development of remote laboratories for Internet-based engineering education in addition to his Ph. D. project that was related to the analysis of electrical machines under particular transient condition. Currently, he is involved on research projects dealing with power optimisation for modern mobile multimedia systems and with Hybrid Electric Energy Storage

**Abderrazek JEMAI** received an Engineer degree from the University of Tunis, Tunisia in 1988 and the DEA and "Doctor" degrees from the University of Grenoble, France, in 1989 and 1992, respectively, all in computer sciences. From 1989 to 1992 he prepared his thesis on simulation of RISC processors and parallel architectures. Since 1993, his interests are focused on high level synthesis and simulation at behavioral and system levels within AMICAL and COSMOS at TIMA Laboratory in Grenoble. Dr Jemai became an assistant professor at the ENSI university in Tunis in 1993 and a Maitre-Assistant Professor at the INSAT University in Tunis since 1994. He was the principal investigator for the "Synthesis and Simulation of VLSI circuits" project at the ENSI/Microelectronic group. He was the principal investigator of the simulation module in AMICAL at TIMA in Grenoble. He is also the principal investigator for the "Performance evaluation of MPSoC" project in LIP2/FST Laboratory in Tunis. Dr Jemai is also working on Security embedded systems.

**Mohamed ABID** is currently professor at Sfax University in Tunisia. He holds a Diploma in electrical engineering in 1986 from the University of Sfax in Tunisia and received his PhD degree in computer engineering in 1989 at University of Toulouse in France. His current research interests include Hardware-Software System on Chip co-design, reconfigurable FPGA, real time system and embedded system. Dr. Abid has authored/co-authored over 100 papers in international journals and conferences. He served on the technical program committees for several international conferences. He also served as a co-organizer of several international conferences.