# High Level H.264/AVC Video Encoder Parallelization for Multiprocessor Implementation

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

[1] Department of Electrical Engineering, CES Laboratory, ENIS Institute, Sfax University, Tunisia, hajer_kri@yahoo.co.nz
[2] LIP2 Laboratory, Faculty of Science of Tunis, Tunisia, Abderrazak.Jemai@insat.rnu.tn
[3] Unité de recherche en Matériaux Mesures et Applications (MMA) INSAT, chiheb.ammari@insat.rnu.tn

***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 a big challenge. For an efficient implementation, we motivate the use of multiprocessor platforms for the execution of a parallel model of the encoder. In this paper, we propose a high-level independent target-architecture parallelization methodology for the development of an optimized parallel model of a H.264/AVC encoder (i.e. a processes network model balanced in communication and computation workload).**

## I. 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. For such a video encoder, it may be probably necessary to use some kind of multiprocessor approach to share the encoding application execution time between several processors.

In this paper, we propose a high-level independent target-architecture parallelization methodology of the H.264/AVC encoder based on the use of parallel programming models of computation. In this methodology, the two predominant concepts of parallelism; the data-level partitioning and the task-level splitting and merging are used. The objective is the exploration of task and data levels parallelism, with the use of communication and computation workload analysis to get an optimal high-level parallel model of the H.264/AVC encoder.

Starting with the H.264 encoder block diagram and using the task-level decomposition, a first parallel model of the encoder will be proposed. This model is based on the Kahn Process Network (KPN) [4] model of computation implemented by the Y-chart Applications Programmers Interface (YAPI) C++ library [5]. Using the starting parallel model, communication and computation workload analysis shall be considered to identify the potential bottlenecks and thus to provide a global guidance when optimizing concurrency between processes. Based on the obtained results, task-merging and data-partitioning are then explored to get an optimized parallel YAPI/KPN model. The goal of this optimization is to get finally a parallel model with the best computation and communication workload balance.

The outline of the paper is as follow. Section 2 defines the adopted experimental environment. In section 3, we present the starting parallel KPN model obtained by task-level decomposition and the results of its system level functional validation. Section 4 discusses the concurrency optimization strategy of the starting parallel YAPI/ KPN model using the task-merging and the data-partitioning forms of parallelism. Finally, section 5 concludes the paper.

## II. EXPERIMENTAL ENVIRONMENT

For the parallel specification of the H.264/AVC encoder, the JM10.2 [6] software reference version is used with main profile @ level 4. The high system-level functional simulation of the obtained parallel models has been performed on a General-Purpose Processor (GPP) platform based on an INTEL Centrino 1.6 GHz running a Linux operating system. For the video streaming and video conferencing applications, we used popular video test sequences in the Quarter Common Intermediate Format (QCIF, 176×144 picture elements).

For an optimal balance between the encoding efficiency and the implementation cost, a proper use of the H.264/AVC tools has been proposed in a previous work [7] to maintain an acceptable performance while considerably reducing complexity. The obtained optimal encoding tools have been fixed as follows. An UMHexagonS fast motion estimation scheme, a search range of 8, 4 variable block sizes from 16x16 to 8x8, 3 reference frames, R-D Lagrangian optimization activated, Hadamard transform disabled, motion vector fractional pixel accuracy enabled, a QP value fixed to 28, and a CAVLC entropy encoding technique. In addition, the H.264/AVC standard uses different encoding structures, including the classical coding types and the advanced pyramid

978-3-9810801-5-5/DATE09 © 2009 EDAA

coding structures. The influence of these coding structures on performance and complexity is also analyzed in [7]. According to the obtained results, the 3Level-5B has been adopted for our fixed optimal configuration.

## III. TASK-LEVEL IMPLEMENTATION OF THE H.264/AVC ENCODER

The goal of this step is to extract the available task-parallelism by splitting compute nodes as far as possible to get the first starting valid parallel KPN model of the encoder. This model will be implemented using the YAPI multi-threading programming environment. The YAPI implemented parallel model is then validated using high level functional simulations. In this section, we will first present the selected communication granularity level and the starting parallel KPN model proposed using the task-level decomposition. After that, the adopted YAPI implementation strategy of the starting parallel model is presented. Finally, the YAPI system level functional validation results are discussed.

### *A. Communication granularity*

For many previous task-level parallelization works [8, 9], the GOP, slice or frame level communication granularity has been used. It has been shown that the GOP granularity level provides the best encoding performance. However, for embedded System-on-Chip implementation, the available memory is limited. Using these systems, the GOP or frame level communication granularity is not viable. For example, if the frame level granularity is selected, the associated FIFO communication channels should have at least one frame size. For low video resolutions (like the QCIF format), a minimum of 38 Kilo bytes FIFO size is needed. For higher HD resolutions, around 3 Mega bytes are necessary to ensure an inter task FIFO communication. This is not practical for limited resources embedded systems.

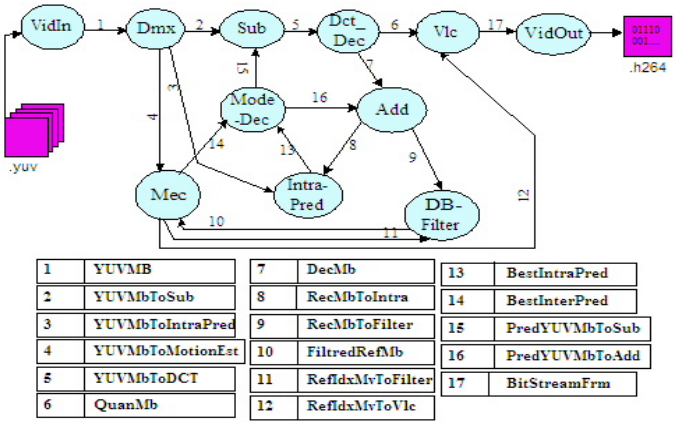
The optimal communication granularity is thus the fine grain level, i.e. at MB level since only the current and reference frames needs to be stored. Each frame is considered as the current workload, and the encoding process of each frame is divided between the processors.

### *B. Starting parallel KPN model*

The Task Level Parallelism (TLP) is first considered [10]. This type of parallelism is achieved by decomposing the whole application into separate blocks. Each block defines one single task or process that runs a separate stage of an algorithm. For this case, the application blocks diagram [1] has served as a starting point for extracting the task-level parallelism. Given this, the sequential H.264/AVC encoding algorithm is first split into concurrent tasks that may be executed at the same time, and then the necessary inter-task communication is established using message passing KPN primitives [4].

Given the functional blocks diagram of the encoder and the sequential C-code specification, the starting proposed model is presented in the following figure 1.

The "VidIn" process shown in figure 1 represents the input of the encoder. It 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" macro-blocks of 16x16 pixels. The "Dmx" process forwards these macro-blocks to the "Sub", "Mec", and "Intra-Pred" processes.




| | | | | | |
|---|---|---|---|---|---|
| 1 | YUVMB | 7 | DecMb | 13 | BestIntraPred |
| 2 | YUVMbToSub | 8 | RecMbToIntra | 14 | BestInterPred |
| 3 | YUVMbToIntraPred | 9 | RecMbToFilter | 15 | PredYUVMbToSub |
| 4 | YUVMbToMotionEst | 10 | FiltredRefMb | 16 | PredYUVMbToAdd |
| 5 | YUVMbToDCT | 11 | RefIdxMvToFilter | 17 | BitStreamFrm |
| 6 | QuanMb | 12 | RefIdxMvToVlc | | |

Figure 1. Starting parallel H.264 KPN model

The "Sub" process reads the predicted "PredYUVMbToSub" macro-block, subtracts it from the current "YUVMbToSub" macro-block, and sends the residual data "YUVMbToDCT" to the "Dct_Dec" process to perform associated transforms, respectively on the Y luminance and the UV chrominance macro-blocks (MBs). These MBs are first arranged into blocks of 4x4 pixels. Each 4x4 block is first transformed into DCT coefficients block 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" process receives the quantized DCT coefficients "QuanMb", performs the CAVLC entropy coding and transmits the resulting "BitStreamFrm" compressed bit stream to the "VidOut" process. Finally, the "VidOut" process sends the H.264 compressed data bit stream to the output file (.h264).

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) macro-bock. Using the current MB "YUVMbToIntraPred" output of the "Dmx" process and the reconstructed previously encoded MB "RecMbToIntra", the "Intra_Pred" process maintains first the storage of this "RecMbToIntra" MB in the reconstructed frame declared as local variable in the "Intra_Pred" process code. Then, it performs an intra-prediction on each macro-block 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 macro-block 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 from the "DB_Filter" process output. The "DB_Filter" process receives the reconstructed decoded "RecMbToFilter" macro-blocks (only the MBs used as reference) from the "Add" process and information about the references indexes and the motion vectors of this macro-block (already inter-predicted) from the "Mec" process. Then, filtering is applied on each reconstructed previously encoded MB to reduce blocking distortions. The best inter-prediction mode cost obtained along with the corresponding predicted MB are sent as the "BestInterPred" structure to the "Mode_Dec" process. Using the best intra-prediction and inter-prediction modes, the "Mode_Dec" process selects the best optimal "PredYUVMbToSub" predicted macro-block of them and transmits it to the "Sub" process.

### *C. YAPI programming strategy*

For the implementation of the parallel model of figure 1 using the YAPI multi-threading runtime environment, we started with the sequential C reference code of the fixed configuration defined in section 3. 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 for this purpose is not allowed. 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.

In the original C source code [6], there are particular specialized functions that are associated to several processes. For example, this is the case of the Rate-Distortion Optimization (RDO) technique that has been activated in our used encoding configuration tools. Setting this RDO option, some specialized functions in the "Vlc" entropy coding process are also used by the "Intra-Pred" and the "Mec" processes. Effectively, to select the best intra and inter prediction modes using the RDO optimization criterion, some VLC functions (like "writeMBLayer()") for computing the rate (number of bits consumed), thus the cost of every possible coding mode are required. For this case, we opted for a redundant implementation of all the associated VLC functions in the "Vlc", "Mec" and "Intra-Pred" processes to minimize the communication overhead at the cost of a maximum computing burden of these processes.

### *D. High-level functional simulation of the starting parallel KPN/YAPI model*

Based on the proposed YAPI programming strategy, the parallel KPN model of figure 1 is implemented using the YAPI multi-threading programming environment. The YAPI system level functional validation results of the implemented model are presented and discussed in this paragraph.

#### *1) Communication workload analysis:*

The proposed parallel model of figure 1 has been validated at YAPI system level. At this level, when this model is executed, the YAPI "read", "write", and "execute" functions generate information on computation and communication workload of the application. For a QCIF "Bridge close" sequence of 13 YUV frames, the communication workload analysis is obtained, and shown in the figure 2.

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 number of tokens per call is equal to 1 for all the "reading" and "writing" operations (T/W and T/R). 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 macro-blocks of 16x16 pixels. One macro-block is constituted with two 8x8 blocks of chrominance, and one 16x16 block of luminance. For example, given 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 used 13 frames sequence, and given the adopted 3Level-5B pyramid coding structure, only 7 are used as reference frames (1 I-frame, 2 P-frames, and 4 B-frames). These frames are maintained by the "Mec" process after receiving the filtered MBs from the "DB_Filter" process via the "FilteredRefMb" FIFO. These references frames are reconstructed from 693 previously encoded macro-bocks communicated between the "DB_Filter" and the "Mec" processes.

| | 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 |

Figure 2. Communication workload of the starting parallel model

The "Tsize" column of one token represents the size of the data structure communicated per call between two communicating processes. Given the "Tsize" column values of figure 2, it is clear that the communication workload is somewhat unbalanced for this starting computational network. The "VidIn" and "VidOut" processes are used for communication with the external environment. The "VidIn" process is responsible for getting the input video from the input file. The "VidOut" process stores the H.264/AVC compressed data to the output file. These are platform dependent tasks and thus, are not considered in the communication workload analysis. 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.

Typically, to get better communication behavior, the data level parallelism and the task level splitting or merging should be used. Data level parallelism consists in splitting the data communicated over selected channels thus duplicating the associated tasks of the model. Task level merging consists in combining pipelined tasks that are exchanging large data structures. The task level splitting extracts the available task-parallelism by further splitting the compute nodes. The decision on data splitting and task merging or splitting will depend on the computational workload analysis of the network.

*2) Computation workload analysis:*

Typically, tasks will not need the same amount of processing time. Thus, a computational workload analysis is considered. For this purpose, a parallel computational "Gprof" GNU [11] profiling is performed. The obtained results are reported in figure 3 in terms of the CPU time percentage spent in the process execution.

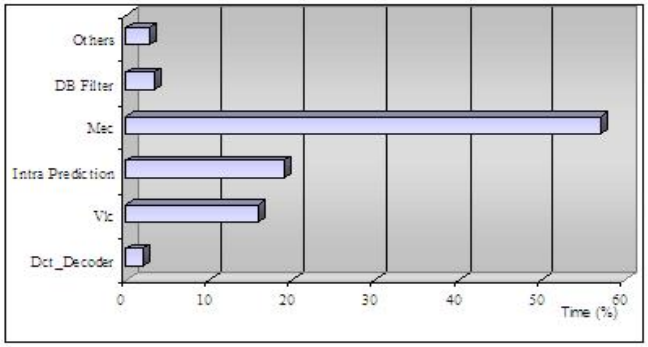



Figure 3. Parallel computational profiling of the first proposed model

Given the profiling results of figure 3, it is clear that the computational workload of the model is too much unbalanced. Some processes have negligible complexity; others especially the "Mec" is still the most computational-expensive task with more than 50% of the total computing time complexity. This is because of the activated Rate-Distortion Optimization (RDO) option which maximizes significantly the coding gain at the cost of a very high computational complexity [12].

Finally it is clear, using the obtained communication and computation workload analysis results, that the starting model of figure 1 does not have good concurrency properties. This outlines the potential of using different steps of task level splitting or merging and data level splitting to derive in a structured way a parallel implementation of the encoder that has a balanced computational workload and good communication behavior.

## IV. DATA-LEVEL SPLITTING OF THE MOTION ESTIMATION AND COMPENSATION PROCESS

This section presents the different steps that have been used to derive in a structured way a parallel implementation of the H.264/AVC encoder that has a balanced workload and good communication behavior. This optimized model has been obtained using the task level merging and data level splitting techniques. The task-merging has been 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 thus removed. For the most computational-expensive "Mec" task, data splitting is proposed for a better concurrency optimization. Given the profiling results of figure 3, the "Mec" process has been split into three "Mec1", "Mec2" and "Mec3" processes. However before deciding for the partitioning of the communicated data to the motion estimation compensation processes, a data dependency analysis applied for these processes has been considered to minimize the dependencies and to maximize the parallelism rate between the triple decomposed tasks.

### *A. Data dependencies analysis in the motion estimation and compensation process*

Several types of data dependencies are introduced in the H.264/AVC standard. In this section, we are concerned only with the data dependencies of the motion estimation and compensation module as the data-partitioning is performed solely for the "Mec" process.

For the inter prediction of a current MB, the "Mec" module requires the left, top, and top-right MBs as shown in Figure 4 (a). In fact, the Predicted Motion Vector (PMV) is first determined using the motion vectors of the neighboring MBs and their corresponding reference indexes. Then, the difference between the final optimal motion vector and the PMV is encoded. On the other hand, the inter-prediction module needs the previously encoded reference frames. Before processing a current MB, the co-located MB and the minimum of its eight neighboring reference frame MBs should be available as shown in figure 4 (b).

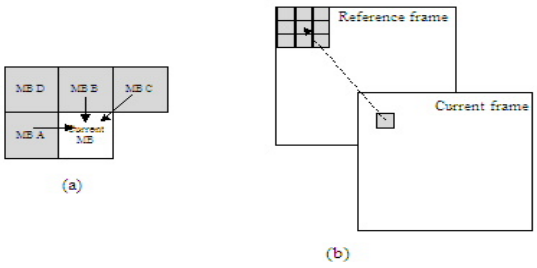



Figure 4. Data dependencies in the inter-prediction module

### *B. Data partitioning strategy*

As mentioned in section 3.1, the communication granularity at the MB-level is selected. The encoding process of each frame is performed MB by MB beginning from the left side. The processing of each frame is thus divided by the different processes of the network. For this, when starting the inter-prediction of a current MB, we have the certitude that the reference data of the co-located MB and their eight neighboring MBs of the previously encoded frame are available in the DPB list of the past encoded frames.

However we are not sure that all the motion vectors of the neighboring MBs of the same frame under processing are already available since each "Mec1", "Mec2", or "Mec3" process will compute separate associated MBs. To minimize the spatial data dependencies between these three inter-prediction "Mec" modules, we propose to split each frame into

three MBs regions. Each region consists in a columns number of MBs, as shown in figure 5. For example as observed in figure 5, the total number of MBs communicated from the "VidIn" process to the "Mec1" and "Mec2" processes is equal to: ((Width/16)/3) * (Height/16) (i.e. 27 MBs for one QCIF resolution low frame).

Tripling the "Mec" process results in tripling the associated input and output FIFOs channels. For example, three "YUVMbToMotionEst" FIFOs are used and so around only the third of the total communication load is transmitted over each FIFO channel. This represents 27*40752 bytes per QCIF frame that are copied MB by MB to the "Mec1" and "Mec2" processes. The token structure has not been modified but only the number of communicated tokens is reduced. For the output FIFOs: "RefIdxMvToFiler", "RefIdxMvToVlc", and "BestInterPred", the same procedure has been used.

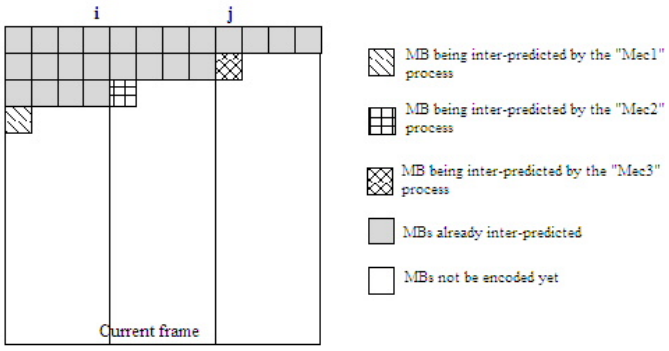



Figure 5. Concurrent inter-predictions at MB-level

### C. *Important implementation issues of the multi motion estimation processes*

The main important problems that we encountered for the effective data-splitting of the three inter-prediction modules are discussed as follows:

#### 1) *Data dependencies between the "Mec$_i$" processes:*

As shown in figure 5, when starting the inter-prediction of the MBs of the column "i" (last column of the associated data "Mec1" process region) from the second line of the frame under processing, the "Mec1" have to receive motion data of the top-right neighboring MBs from the "Mec2" process. On the other hand, to process the MBs of the column "j" (first column of the associated data "Mec3" process region), the "Mec3" process requires the motion data of the left neighboring MBs from the "Mec2" process.

However, before starting the inter-prediction of the MBs of the column "i+1" and these of the column "j-1" starting from the second line, the "Mec2" process should have read the motion data of respectively the left neighboring MBs from the "Mec1" process and the top-right neighboring MBs from the "Mec3" process.

#### 2) *Reference frames memory:*

As it is applied for all the channels connected to the "Mec$_i$" processes, three "FilteredRefMb" FIFOs are used. Typically, it is not possible to split the reference frame between the three used FIFOs since the multi inter-prediction modules need always the same reference frames. However, given the performed data dependencies analysis, it is sufficient to use the co-located MB and its eight neighboring MBs of the corresponding reference frame to inter-predict a current MB. For this, we propose to partition the reference data and thus to send to each "Mec$_i$" process just the needed reference data. For this, we suggest to split each reference frame into three partitions and to transmit each reference data partition to only its corresponding FIFO, as shown in the figure 6.

Once one MB is filtered, it will be copied in one or two associated reference FIFOs. For example as observed in figure 6, the total number of filtered reference MBs, outputs of the "DB_Filter" process and received by the "Mec1" module, is equal to: ((Width/16)/3 + 1) * (Height/16) (i.e. 36 filtered MBs for one QCIF resolution low frame).

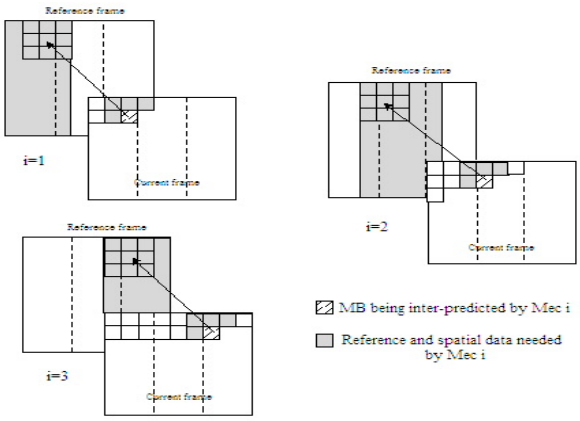


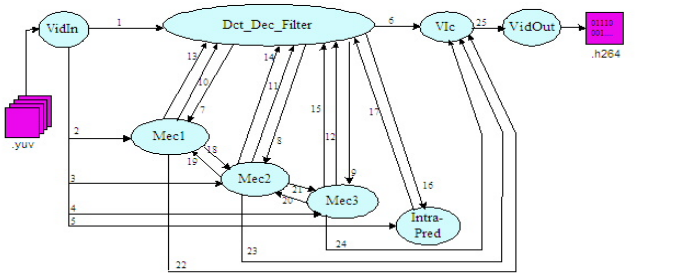

Figure 6. Reference data partitioning between the three inter-prediction modules

### D. *Optimized parallel KPN model of the H.264/AVC encoder*

The optimized parallel model obtained is given in figure 7. 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 "Meci" processes and their environment. This model has been implemented and validated at YAPI system level. The communication workload results are obtained and shown in figure 8 for the same QCIF "Bridge close" 13 frames sequence. A computational "Gprof" profiling is also performed and reported in figure 9.

It is clear from figure 8 that the total number of tokens communicated from/to the motion estimation and compensation processes has been reduced. Effectively, the number of tokens transmitted over the "YUVMBToMotionEst1" connecting the input of the "Mec1" process has been reduced to 324 MBs tokens (27 MBs per frame * 12 P&B frames). The total bytes number communicated over this channel is 40752*27*1 bytes. Among 7 reference frames, only 252 filtered reference MBs tokens of 1032 bytes token size (36MBs * 7) are copied in the "FiteredRefMb1" channel from the "Dct_Dec_Filter" process to the "Mec1" process. Given the obtained "Tsize" column values of figure 8, except the large token data structure

transmitted into the "YUBMBToDCT" FIFO, it is clear that the optimized proposed model has better communication behavior compared to the starting model.




| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | YUVMbToDCT | 9 | FiltredRefMb3 | 17 | BestIntraPred | 25 | BitSteamFrm |
| 2 | YUBMbToMotionEst1 | 10 | RefIdxMvToFilter1 | 18 | RefIdxMvToMec2 | | |
| 3 | YUBMbToMotionEst2 | 11 | RefIdxMvToFilter2 | 19 | RefIdxMvToMec1 | | |
| 4 | YUBMbToMotionEst3 | 12 | RefIdxMvToFilter3 | 20 | RefIdxMvToMec2_2 | | |
| 5 | YUVMBToIntraPred | 13 | BestInterPred1 | 21 | RefIdxMvToMec3 | | |
| 6 | QuanMbToVLC | 14 | BestInterPred2 | 22 | RefIdxMvToVlc1 | | |
| 7 | FiltredRefMb1 | 15 | BestInterPred3 | 23 | RefIdxMvToVlc2 | | |
| 8 | FiltredRefMb2 | 16 | RecMbToIntra | 24 | RefIdxMvToVlc3 | | |

Figure 7. Proposed Optimized Parallel KPN model of the H.264 encoder

| | size | Tsize | Wtokens | Wcalls | T/W | Rtokens | Rcalls | T/R |
|---|---|---|---|---|---|---|---|---|
| h264.YUVMBToDCT | 1184 | 40752 | 1287 | 1287 | 1 | 1287 | 1287 | 1 |
| h264.BestIntraPred | 1 | 648 | 1287 | 1287 | 1 | 1287 | 1287 | 1 |
| h264.YUVMbToMotionEst1 | 319 | 40752 | 324 | 324 | 1 | 324 | 324 | 1 |
| h264.YUVMbToMotionEst2 | 321 | 40752 | 324 | 324 | 1 | 324 | 324 | 1 |
| h264.YUVMbToMotionEst3 | 539 | 40752 | 540 | 540 | 1 | 540 | 540 | 1 |
| h264.QuanMbToVLC | 1 | 40752 | 1287 | 1287 | 1 | 1287 | 1287 | 1 |
| h264.YUVMBToIntraPred | 1182 | 40752 | 1287 | 1287 | 1 | 1287 | 1287 | 1 |
| h264.RecMbToIntra | 1 | 592 | 1287 | 1287 | 1 | 1286 | 1286 | 1 |
| h264.RefIdxMvToFilter1 | 3 | 416 | 162 | 162 | 1 | 162 | 162 | 1 |
| h264.RefIdxMvToVlc1 | 3 | 304 | 324 | 324 | 1 | 324 | 324 | 1 |
| h264.FiltredRefMb1 | 1 | 1032 | 252 | 252 | 1 | 252 | 252 | 1 |
| h264.BestInterPred1 | 3 | 648 | 324 | 324 | 1 | 324 | 324 | 1 |
| h264.RefIdxMvToFilter2 | 3 | 416 | 162 | 162 | 1 | 162 | 162 | 1 |
| h264.RefIdxMvToVlc2 | 3 | 304 | 324 | 324 | 1 | 324 | 324 | 1 |
| h264.FiltredRefMb2 | 2 | 1032 | 315 | 315 | 1 | 315 | 315 | 1 |
| h264.BestInterPred2 | 3 | 648 | 324 | 324 | 1 | 324 | 324 | 1 |
| h264.RefIdxMvToFilter3 | 2 | 416 | 270 | 270 | 1 | 270 | 270 | 1 |
| h264.RefIdxMvToVlc3 | 2 | 304 | 540 | 540 | 1 | 540 | 540 | 1 |
| h264.FiltredRefMb3 | 4 | 1032 | 378 | 378 | 1 | 378 | 378 | 1 |
| h264.BestInterPred3 | 1 | 648 | 540 | 540 | 1 | 540 | 540 | 1 |
| h264.RefIdxMvToMec2 | 1 | 416 | 108 | 108 | 1 | 108 | 108 | 1 |
| h264.RefIdxMvToMec1 | 1 | 416 | 96 | 96 | 1 | 96 | 96 | 1 |
| h264.RefIdxMvToMec2_2 | 1 | 416 | 96 | 96 | 1 | 96 | 96 | 1 |
| h264.RefIdxMvToMec3 | 1 | 416 | 108 | 108 | 1 | 108 | 108 | 1 |
| h264.BitStreamFrm | 12 | 40 | 13 | 13 | 1 | 13 | 13 | 1 |

Figure 8. Communication workload of the optimized parallel model

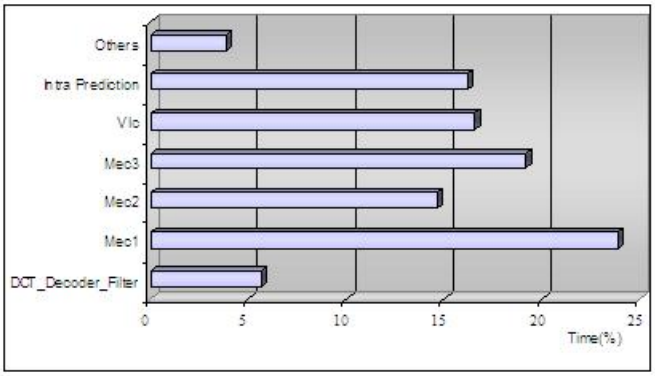



Figure 9. Parallel computational profiling of the final model

In addition, as indicated in figure 9, the data partitioning of the motion estimation and compensation 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 motion estimation and compensation to more reduce the computational workload on the associated processes.

## V. CONCLUSION

For its cost-effective multiprocessor implementation, the H.264/AVC encoder has been parallelized at high system level using a parallel streaming programming model. Starting with the reference C code, a first 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 environment. Using communication and computation workload analysis of the proposed KPN/YAPI model, it is shown that the first proposed parallel model does not have good concurrency properties. Based on these results, different steps of task level merging and data level splitting are used to derive in a structured way a parallel implementation of the H.264/AVC encoder that has a balanced computation workload and good communication behavior.

## REFERENCES


[1] A. Joch, F. Kossentini, P. Nasiopoulos, A Performance Analysis of the ITU-T Draft H. 26L Video Coding Standard, in: Proc. 2002, 12th International Packet Video Workshop, Pittsburg, Pa, USA.

[2] M. Alvarez, A. Salami, A. Ramirez, M. Valero, A Performance Characterization of high Definition Digital Video Decoding using H264/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] G. Kahn, The semantics of a simple language for parallel programming, in: Proc. 1974, the IFIP Congress 74, North-Holland Publishing Co.

[5] 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.

[6] H264 Reference Software Version JM 10.2, http://iphome.hhi.de/suehring/tml/. (November 2005).

[7] H. Krichene, A.C. Ammari, A. Jemai, M. Abid, Performance/Complexity Analysis of a H264 Video Encoder, International Review on Computers and Software (IRECOS), Vol 2 n°4, pp n°401-414, July 2007.

[8] 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.

[9] 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.

[10] M. Pastrnak, P.H.N. de With, S. Stuijk, and J. van Meerbergen, Parallel Implementation of Arbitrary-Shaped MPEG-4 Decoder for Multiprocessor Systems, in: Proc. 2006. Visual Communications and Image Processing (VCIP'06). pp 60771I-1 - 60771I-10.

[11] 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/

[12] F. Pan, H. Yu, Z. Lin, Scalable Fast Rate-Distortion Optimization for H.264/AVC, EURASIP Journal on Applied Signal Processing, Article ID 37175 (Volume 2006), pp.1–10, DOI 10.1155/ASP/2006/371