Systematic analysis of the decoding delay in multiview video

Pablo Carballeira , Julián Cabrera, Fernando Jaureguizar, Narciso García

A B S T R A C T

We present a framework for the analysis of the decoding delay in multiview video coding (MVC). We show that in real-time applications, an accurate estimation of the decoding delay is essential to achieve a minimum communication latency. As opposed to single-view codecs, the complexity of the multiview prediction structure and the parallel decoding of several views requires a systematic analysis of this decoding delay, which we solve using graph theory and a model of the decoder hardware architecture. Our framework assumes a decoder implementation in general purpose multi-core processors with multi-threading capabilities. For this hardware model, we show that frame processing times depend on the computational load of the decoder and we provide an iterative algorithm to compute jointly frame processing times and decoding delay. Finally, we show that decoding delay analysis can be applied to design decoders with the objective of minimizing the communication latency of the MVC system.

1. Introduction

For several years, video technologies have targeted the development of systems that provide immersive viewing experiences. Nowadays, the advances in three-dimensional (3D) display technologies have made 3D video an emerging and sustainable market in the near future. 3D Video (3DV) and free viewpoint video (FVV) are new types of visual media that expand the user's experience beyond what is offered by 2D video [1], providing a 3D depth impression of the scene, and interactive viewpoint selection. Currently, these types of visual media systems are beginning to enter into consumer markets, such as entertainment and mobile applications [2]. For those systems, a data format that is richer than single 2D video signal is needed. The spectrum of data formats for 3D Video goes from purely image-based data formats like multiview video (multiple views of the same scene) to data formats related to computer graphics like 3D meshes and their corresponding textures [3]. A widely adopted approach is the one that includes multiview video and depth sequences as additional scene geometry information, allowing the possibility of generating additional views on virtual camera positions [4]. Nevertheless, the size of this multiview video grows linearly with the number of views while the available bandwidth is generally limited. Thus, an efficient compression scheme for multiview video is needed.

Multiview video coding (MVC) [5] is an extension of the H.264/MPEG-4 Advanced Video Coding (AVC) standard [6] that provides efficient coding of such multiview video. Besides, as depth signals can be represented as monochromatic video signals, MVC has been also commonly used to compress them [4]. As an extension of AVC, MVC makes use of the set of AVC coding tools. The key additional feature of the MVC design, that increases the coding efficiency specifically for multiview video, is a new prediction relationship between frames of different views that exploits the interview redundancy. This prediction relationship is known as interview prediction. Fig. 1 shows a sample prediction structure in which temporal and interview predictions are used.

MVC allows a wide range of applications and scenarios [7]. Here, we address real-time applications such as live broadcasting, videoconferencing or interactive streaming [8] where constraints on the end-to-end delay are imposed. The one-way delay between both ends of the conversation is known as *communication latency*, i.e., the delay between the instant when a frame is captured and the instant when it is displayed at the receiver. In bidirectional applications, the constraint on communication latency is stricter. For those, typical recommendations on maximum communication latency generally state that there is none or little impact below 150 ms, while a serious impact may be observed above 400 ms [9].

Each element (encoder, transmission channel and decoder) contributes to the delay between the instant when a frame is captured and the instant when it is decoded at the receiver: the *system delay*. For each frame, the value of the system delay varies due to different factors, such as the required encoding time or the nature of the transmission channel (variable or constant bitrate, packet losses, etc.). Since frames have to be displayed at a constant rate, generally receivers utilize an output buffer for decoded frames, to guarantee a constant communication latency. In practice, this buffer results in an additional variable delay for each decoded frame: the *display delay*. Therefore, the communication latency is the sum of the system delay and the display delay. In real-time applications, the design of

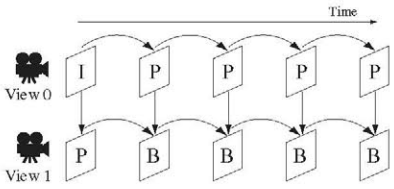


Fig. 1. Example of a multiview prediction structure for two cameras. Horizontal arrows correspond to temporal prediction and vertical arrows to interview prediction.

this output buffer, and the display delay is a challenging issue. On the one hand, the display delay should be as minimum as possible since it increases the communication latency. On the other hand, it has to be high enough to absorb the variability of the system delay so that frames are displayed at a constant rate. While in non-live services, such as video on demand, this display delay may be over-dimensioned with little impact on the service, this requirement is stricter in the case of real-time bidirectional services. Thus, an accurate computation of the system delay is essential to design a system with a minimum valid display delay.

In [10], we presented a framework for the analysis of the encoding delay for MVC. Now, in this paper, we focus on the analysis of the contribution of MVC decoders to the system delay: the *decoding delay*. Our purpose here is to provide tools for an accurate evaluation of the decoding delay in order to complete the analysis of the contribution of the MVC codec processes to the system delay. The decoding delay in MVC decoders depends on two different but related factors:

1. The multiview prediction structure: temporal and interview prediction relationships among frames establish decoding order dependencies for a frame.
2. The hardware architecture and implementation of the decoder: specific architectural features of multiview decoders (e.g. number of processors, use of threads etc.) influence the time needed to decode a given frame, and therefore, they affect the decoding delay performance.

Whereas in single view decoders, the computation of the decoding delay can be easily approximated as the decoding time of one frame, in the case of MVC, the complexity of multiview prediction structures, and the presence of several views that need to be decoded simultaneously, increase the complexity of the decoding delay analysis. Thus, we present here a framework for the systematic analysis of the decoding delay in MVC decoders. This framework evaluates the decoding delay taking into account: (i) the multiview prediction structure and (ii) the hardware implementation of the decoder. Nowadays, actual decoders support several parallel streams and different codecs, and the general tendency is to incorporate general purpose processors, in which the decoders are software-implemented, instead of traditional dedicated hardware processors. This tendency is particularly interesting to handle MVC streams due to its inherent parallelization characteristics [11,12]. Therefore, our framework assumes a hardware platform for the decoder based on a multi-core processor with multi-threading capabilities. We define a *decoding process* as the set of operations that are needed to decode a frame. Our model assumes that any decoding process can run on an exclusively dedicated core (processor from now on) or one of the threads that share the processing power of one of the processors. The required time to run that process will be higher if several processes share the same processor.

Analogously to the encoding latency analysis [10], we rely on graph theory to compute the decoding delay for this hardware model. A graph is constructed from the multiview prediction structure in which the frames can be seen as the nodes and the prediction dependencies as the edges. Each edge has an associated cost that represents the contribution of the prediction dependency to the decoding delay. We show that frame processing times depend on the computational load of the decoder and we provide an iterative algorithm to compute jointly frame processing times and the decoding delay by an iterative analysis of the graph.

In our results, we use the decoding delay analysis to characterize the communication latency of a complete MVC system. We show that this analysis can be used to determine hardware requirements of MVC decoders, such as number of processors or processor throughput (number of frames that one processor is able to decode per second), with the objective of achieving a target communication latency. For example, we show that for a given processor throughput, the decoding delay can be reduced by increasing the number of processors in the decoder, until certain limit that we can identify. Increasing the number of processors above that limit does not further decrease the decoding delay.

This paper is organized as follows: in Section 2, we discuss the communication latency of an MVC system and the role of the decoding delay on it. In Section 3, we present our framework for the decoding delay analysis in a multi-thread decoder architecture. In Section 4 we present the iterative algorithm for the computation of processing times and decoding delay. In Section 5 we show the experimental results and in Section 6 we present the conclusions.

2. Discussion on communication latency of MVC systems

As aforementioned, the communication latency indicates the time elapsed between the instant when a frame is captured, and the instant when that frame is displayed. A block diagram of an MVC system and the elements that add to the communication delay between its both ends, are depicted in Fig. 2. For frame x_j^i (frame j of view i), $t_{\text{capt}_j^i}$ is the instant when x_j^i is captured, $t_{\text{cod}_j^i}$ is the time instant when x_j^i is completely coded, $t_{\text{RX}_j^i}$ is the instant when the

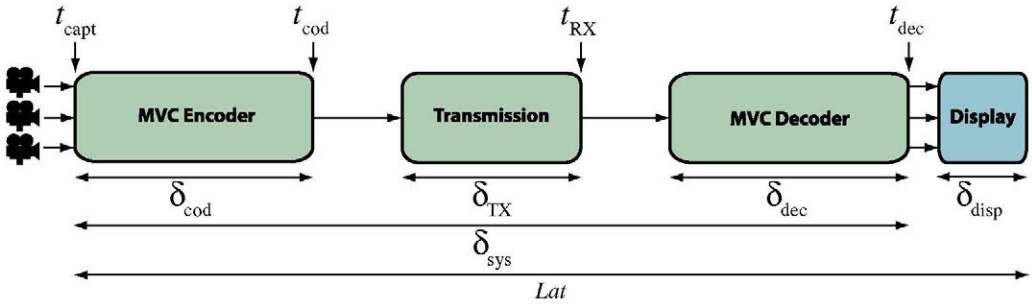


Fig. 2. Block diagram of the MVC system with encoding delay, transmission delay, decoding delay, system delay, display delay and communication latency.

coded version of x_j^i is received at the decoder and $t_{\text{dec}_j^i}$ is the instant when x_j^i is completely decoded. With this, the encoder delay $\delta_{\text{cod}_j^i}$ is:

$$\delta_{\text{cod}_j^i} = t_{\text{cod}_j^i} - t_{\text{capt}_j^i}, \tag{1}$$

the transmission delay $\delta_{\text{TX}_j^i}$ is:

$$\delta_{\text{TX}_j^i} = t_{\text{RX}_j^i} - t_{\text{cod}_j^i}, \tag{2}$$

and the decoding delay $\delta_{\text{dec}_j^i}$ is:

$$\delta_{\text{dec}_j^i} = t_{\text{dec}_j^i} - t_{\text{RX}_j^i}. \tag{3}$$

We define the system delay $\delta_{\text{sys}_j^i}$ as the time elapsed between the capture time of x_j^i and the instant when it is completely decoded in the receiver. Formally, it can be expressed as:

$$\delta_{\text{sys}_j^i} = \delta_{\text{cod}_j^i} + \delta_{\text{TX}_j^i} + \delta_{\text{dec}_j^i}. \tag{4}$$

This system delay is variable for each frame, due to the variability of the encoding and decoding delays introduced by the characteristics of the encoding process, such as different types of frame, the prediction structure, etc., and the variable nature of the delay on transmission channels. In order to maintain a constant display frame rate, the communication latency must have a constant value for all the frames. Therefore, to absorb the variability of the delay added by previous blocks, the receiver uses a buffer for decoded frames, that adds a display delay, $\delta_{\text{disp}_j^i}$, which is variable for each frame. Thus, in practice, the communication latency can be expressed as:

$$Lat = \delta_{\text{sys}_j^i} + \delta_{\text{disp}_j^i} = \delta_{\text{cod}_j^i} + \delta_{\text{TX}_j^i} + \delta_{\text{dec}_j^i} + \delta_{\text{disp}_j^i}. \tag{5}$$

In real-time applications, the optimal communication latency should be the minimum delay that allows the receiver to maintain a constant display rate. This latter condition means that the communication latency value cannot be lower than the system delay of any frame, since this would lead to assign a frame a negative display delay. That is, for a given MVC system, valid communication latency values have to fulfill the following condition:

$$\delta_{\text{disp}_j^i} = Lat - \delta_{\text{sys}_j^i} \geqslant 0, \forall i,j. \tag{6}$$

Therefore, to achieve the minimum valid communication latency value, the following condition must hold:

$$Lat_{\min} - \max_{\substack{i=0,\ldots,N-1 \\ j=0,\ldots,M-1}} \left(\delta_{\text{sys}_j^i}\right) = 0, \tag{7}$$

where N is the number of views and M is the number of frames per view. If the condition in (7) holds, the decoder does not add any display delay to the frame with the highest system delay. Formally:

$$Lat_{\min} = \max_{\substack{i=0,\ldots,N-1 \\ j=0,\ldots,M-1}} \left(\delta_{\text{sys}_j^i}\right) = \max_{\substack{i=0,\ldots,N-1 \\ j=0,\ldots,M-1}} \left(\delta_{\text{cod}_j^i} + \delta_{\text{TX}_j^i} + \delta_{\text{dec}_j^i}\right). \tag{8}$$

Therefore, identifying the frame with highest system delay and its accurate value is an essential factor to obtain the minimum communication latency for well-designed real-time applications. In the next section, we present the framework for the characterization of one of the elements that contribute to that system delay, the decoding delay $\delta_{\text{dec}_j^i}$, in order to complete the contribution of MVC codecs to the communication latency.

3. Analysis of decoding delay in MVC

In this section, we present the elements and the algorithms for the systematic analysis of the decoding delay. Firstly, we discuss the general time relationships in the decoder that allow us to evaluate the decoding delay. Secondly, we present the main elements to compute the decoding delay for any multiview prediction structure in a multi-processor decoder: (i) the parallel multi-processor decoder model and (ii) the direct acyclic graph model. In the next section we present the iterative algorithm to compute jointly the frame processing times and the decoding delay in that decoder architecture.

In order to develop a systematic analysis of the decoding delay on multiview decoders, we make the following assumptions:

1. All views have to be decoded, as all of them will be displayed or the receiver will be able to choose any view for displaying among those received at any time.
2. A frame is the basic decoding unit and is decoded sequentially, i.e., different decoding operations for a given frame cannot be performed in parallel at the same time in several processors.
3. The decoding of a new frame does not start until its reference frames have been completely decoded.

The decoding delay, $\delta_{\text{dec}_j^i}$, as defined in (3), is the difference between the instant when x_j^i arrives at the decoder, $t_{\text{RX}_j^i}$, and the instant when x_j^i is already decoded, $t_{\text{dec}_j^i}$. From (1)–(3), $t_{\text{RX}_j^i}$ can be computed as:

$$t_{\text{RX}_j^i} = t_{\text{capt}_j^i} + \delta_{\text{cod}_j^i} + \delta_{\text{TX}_j^i}, \tag{9}$$

where the capture time $t_{\text{capt}_j^i}$ is known, $\delta_{\text{cod}_j^i}$ can be estimated using [10], and $\delta_{\text{TX}_j^i}$ can be estimated with a transmission channel model [13].

Regarding the computation of $t_{\text{dec}_j^i}$, we define $t_{\text{start}_j^i}$ as the instant when the decoding process of x_j^i starts. Then:

$$t_{\text{dec}_j^i} = t_{\text{start}_j^i} + \Delta t_{\text{proc}_j^i}, \tag{10}$$

where $\Delta t_{\text{proc}_j^i}$ is the processing time devoted to decoding the coded version of x_j^i. We also need to define another relevant time instant, $t_{\text{ready}_j^i}$, as the instant when x_j^i is ready for being decoded, i.e., all its reference frames have been completely decoded. $t_{\text{ready}_j^i}$ is computed as follows:

$$t_{\text{ready}_j^i} = \max\left(t_{\text{RX}_j^i}, \max_{l \in L(i,j)}(t_{\text{dec}_l})\right), \tag{11}$$

where $L(i,j)$ is the set of reference frames for x_j^i.

While (3), (10) and (11) only depend on the coding order relationships imposed by the prediction structure, and therefore they are valid for all hardware decoder architectures, the relationship between $t_{\text{start}_j^i}$ and $t_{\text{ready}_j^i}$ depends on the specific hardware decoder architecture being used (e.g., number of processors, sequential or parallel processing, etc.). Nevertheless, for any hardware decoder architecture, if we assume that a given frame cannot be decoded before its reference frames have been decoded, then:

$$t_{\text{start}_j^i} \geqslant t_{\text{ready}_j^i}. \tag{12}$$

The decoding process of x_j^i cannot start until all frames in $L(i,j)$ have been decoded, but the start of the decoding of x_j^i may be delayed if there are not processing resources available at $t_{\text{ready}_j^i}$. Thus, the analysis has to be individualized for a given decoder hardware platform.

3.1. Parallel multi-processor decoder model

Due to the inherent parallel characteristics of MVC, parallelization is an essential factor for an efficient implementation of MVC decoders. This may be done by the utilization of multi-core processors and/or parallelization techniques such as multi-threading.

As the use of software decoders implemented in general purpose multi-core processors has grown in recent years, we propose a decoder model that simulates the characteristics of those decoders. We name it parallel multi-processor decoder (Parallel MPD) model. It considers a set of K processors with multi-task decoding (one processor can decode several frames at a time, by means of parallelization techniques).

The characteristics of the Parallel MPD model are the following:

1. The decoding operations for any frame from any of the N views can be performed in any of the K processors.
2. The processors can decode their assigned frames in a parallel way, i.e., if at a given time all the processors are busy and a new frame is ready to be decoded, its decoding process starts immediately in one of the processors in parallel with the current ongoing processes.
3. The decoder manages the assignment of each of the decoding processes to each of the available processors.

3.2. Directed acyclic graph model

In decoder architectures with multitask processors, such as the Parallel MPD model, the decoder assigns the decoding of new received frames to one of the processors without having to wait for the availability of idle processors. Thus, decoding of x_j^i starts at $t_{\text{ready}_j^i}$. Formally:

$$t_{\text{start}_j^i} = t_{\text{ready}_j^i}. \tag{13}$$

With this condition and (11):

$$t_{\text{start}_j^i} = \max\left(t_{\text{RX}_j^i}, \max_{l\in L(i,j)}(t_{\text{dec}l})\right), \tag{14}$$

and using (10):

$$t_{\text{dec}_j^i} = \max\left(t_{\text{RX}_j^i}, \max_{l\in L(i,j)}(t_{\text{dec}l})\right) + \Delta t_{\text{proc}_j^i}. \tag{15}$$

Under the condition in (13), (15) can be solved using a similar approach to that one in [10] that relies on graph theory. In the following we describe it.

For any feasible MVC prediction structure, we can extract a directed acyclic graph (DAG) [14], in which the frames are the nodes of the DAG and the prediction dependencies are its edges. Due to the directed nature of the dependencies (one frame is predicted from the reference frame but not vice versa), the graph is directed. Each directed edge links a reference node (parent) to the node that is predicted from it (child). A path is a sequence of nodes linked by directed edges. Fig. 3 shows an example prediction structure and its associated DAG.

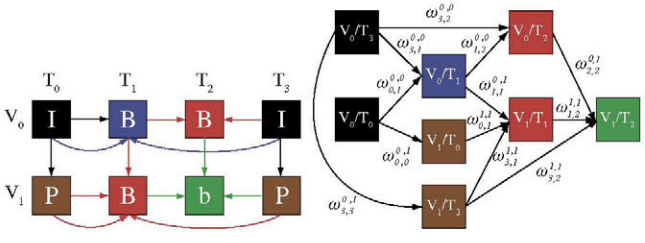


Fig. 3. Example of multiview prediction structure and its associated DAG. Nodes of the DAG represent frames while edges represent dependency relationships in the prediction structure. $\omega_{j,l}^{i,k}$ is the cost of each edge.

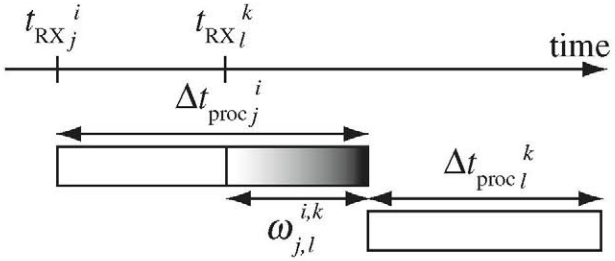


Fig. 4. Graphic significance of the cost value of the edges in the graph.

Each edge of the DAG has an associated cost value that indicates the single contribution of its parent node to the decoding delay of its child node. The cost value $\omega_{j,l}^{i,k}$ of the edge that links x_j^i with x_l^k is:

$$\omega_{j,l}^{i,k} = \max\left(0, \left(t_{\text{RX}_j^i} + \Delta t_{\text{proc}_j^i}\right) - t_{\text{RX}_l^k}\right). \tag{16}$$

In (16) we assume that x_j^i starts its decoding process at $t_{\text{RX}_j^i}$ (the start of the decoding process of x_j^i is not delayed by the decoding processes of its parent frames) to capture the isolated contribution of x_j^i to the decoding delay of x_l^k. As only positive delay values have a realistic meaning, $\omega_{j,l}^{i,k}$ is restricted to positive values. Fig. 4 illustrates the computation of $\omega_{j,l}^{i,k}$ with a time chronogram in which the decoding process of the parent frame x_j^i delays the decoding start of child frame x_l^k. Note that x_l^k is received at $t_{\text{RX}_l^k}$, but its encoding process cannot start until x_j^i is not completely decoded.

The cost of a path is the sum of the costs of the edges that link the nodes in the path. The cost of the path that ends on a given frame sums up the contributions of all parents frames on that path to the decoding delay of that frame. Among the set of paths ending on the same node x_j^i, we name *delay path* to x_j^i, to the one with the highest cost value. Its associated cost, $p_{\text{del}_j^i}$ is the following:

$$p_{\text{del}_j^i} = \max_{u\in U}\{p_j^i(u)\}, \tag{17}$$

where U is the set of paths ending in node x_j^i and $p_j^i(u)$ is the cost of the u^{th} path. $p_{\text{del}_j^i}$ indicates the elapsed time between $t_{\text{RX}_j^i}$ and $t_{\text{ready}_j^i}$. Formally:

$$t_{\text{ready}_j^i} = t_{\text{RX}_j^i} + p_{\text{del}_j^i}. \tag{18}$$

Given that, and the condition in (13), (10) becomes:

$$t_{\text{dec}_j^i} = t_{\text{RX}_j^i} + p_{\text{del}_j^i} + \Delta t_{\text{proc}_j^i}, \tag{19}$$

and therefore the decoding delay is:

$$\delta_{\text{dec}_j^i} = p_{\text{del}_j^i} + \Delta t_{\text{proc}_j^i}. \tag{20}$$

Therefore, provided that the values of $\Delta t_{\text{proc}_j^i}$ are known, $\delta_{\text{dec}_j^i}$ and the decoding chronogram can be computed systematically for all the frames using the DAG.

Table 1
Estimation of frame processing times ($\Delta t_{\text{sim1}}^{\text{I}}$, $\Delta t_{\text{sim1}}^{\text{P}}$ and $\Delta t_{\text{sim1}}^{\text{B}}$) and the parameters α_{P} and α_{B} for different video sequences. Spatial resolutions: ballroom [15] 640 × 480, Newspaper [16] and Balloons [17] 1024 × 768.

Sequence	$\Delta t_{\text{sim1}}^{\text{I}}$ (ms)	$\Delta t_{\text{sim1}}^{\text{P}}$ (ms)	$\Delta t_{\text{sim1}}^{\text{B}}$ (ms)	α_{P}	α_{B}
Ballroom	14.04	8.75	10.38	0.62	0.74
Newspaper	32.58	18.24	24.74	0.56	0.76
Balloons	31.05	21.81	27.84	0.70	0.89

Table 2
Estimation of frame processing times for different numbers of frames decoded simultaneously in a processor ($\Delta t^{(\cdot)}_{\text{sim1}}$ to $\Delta t^{(\cdot)}_{\text{sim4}}$) and ratios of processing time increase for parallel processing.

Frame type	$\Delta t^{(\cdot)}_{\text{sim1}}$ (ms)	$\Delta t^{(\cdot)}_{sim2}$ (ms)	$\Delta t^{(\cdot)}_{sim3}$ (ms)	$\Delta t^{(\cdot)}_{\text{sim4}}$ (ms)	$\Delta t^{(\cdot)}_{sim2}/\Delta t^{(\cdot)}_{\text{sim1}}$	$\Delta t^{(\cdot)}_{sim3}/\Delta t^{(\cdot)}_{\text{sim1}}$	$\Delta t^{(\cdot)}_{\text{sim4}}/\Delta t^{(\cdot)}_{\text{sim1}}$
I	32.13	63.16	91.65	123.82	1.97	2.85	3.85
P	20.81	40.59	58.93	77.75	1.95	2.83	3.74
B	26.45	52.69	77.40	99.76	1.99	2.93	3.77

3.3. Frame processing time model

In our previous work [10,18], we used a frame processing time model for an MVC encoder in which the time devoted to encoding a given frame depends on the number of reference frames. However, in the case of the decoder we assume that the time devoted of the decoding process of a frame depends on the frame type, i.e., I, P or B.

We define the computational load of the decoding process of a frame as the processing time devoted to decoding that frame in an exclusively dedicated processor ($\Delta t^{(\cdot)}_{\text{sim1}}$, where $(\cdot)$ can be I, P or B). We take as a reference the computational load of the decoding process of an I-frame ($\Delta t^{\text{I}}_{\text{sim1}}$) as non extra motion compensation operations are involved in the decoding process. Then, in our model, we consider that the computational load of the decoding process of a P-frame, $\Delta t^{\text{P}}_{\text{sim1}}$, and a B-frame, $\Delta t^{\text{B}}_{\text{sim1}}$, are proportional to $\Delta t^{\text{I}}_{\text{sim1}}$ and computed as follows:

$$\begin{aligned}\Delta t^{\text{P}}_{\text{sim1}} &= \alpha_{\text{P}} \Delta t^{\text{I}}_{\text{sim1}} \\ \Delta t^{\text{B}}_{\text{sim1}} &= \alpha_{\text{B}} \Delta t^{\text{I}}_{\text{sim1}},\end{aligned} \tag{21}$$

where α_{P} and α_{B} are scalar values.

To estimate the parameters in our frame processing time model ($\Delta t^{\text{I}}_{\text{sim1}}$, $\Delta t^{\text{P}}_{\text{sim1}}$, $\Delta t^{\text{B}}_{\text{sim1}}$, α_{P} and α_{B}), we have performed a series of experiments using a JMVC 8.5 decoder [19] running in a general purpose PC: four-core processor working at 2.40 GHz, with 3.25 GB of RAM memory. Using this reference software decoder we have estimated the average decoding time for each type of frame when each frame is decoded in an exclusively dedicated core. From those results we have computed the values of α_{P} and α_{B}. The results for several tested sequences are shown in Table 1. It can be seen that the value of α_{P} and α_{B} depends on the video sequence. α_{P} varies from 0.56 to 0.70 and α_{B} varies from 0.74 to 0.89.

The translation of the computational load to time devoted to decoding a frame, $\Delta t_{\text{proc}^i_j}$, clearly depends on the hardware characteristics of the MVC decoder. For multi-task processors such as the ones in the Parallel MPD model, $\Delta t_{\text{proc}^i_j}$ depends on the computational load conditions of the set of processors. Thus, if a processor has to decode a single frame, its processing time is: $\Delta t_{\text{proc}^i_j} = \Delta t^{(\cdot)}_{\text{sim1}}$. Otherwise, if the processor has to deal with several frames in parallel, the processing time of each frame will increase, i.e., $\Delta t_{\text{proc}^i_j} > \Delta t^{(\cdot)}_{\text{sim1}}$. For a given decoder with K processors, parallel decoding will occur when the number of frames simultaneously decoded at a given time, n_{sim}, is higher than K. $\Delta t_{\text{proc}^i_j}$ also depends on the how the decoder manages the assignment of frames to the available processors. For instance, in a decoder with $K = 2$ and $n_{\text{sim}} = 3$, $\Delta t_{\text{proc}^i_j}$ would differ if: (a) the three frames are decoded in different threads of only one processor or (b) two frames are decoded in two threads in one of the processors and the other frame is decoded in the other processor.

In our model we assume the following assumption: if n_{sim} frames are decoded simultaneously in one processor, the required processing time to decode those frames is n_{sim} times the computational load of those frames. Formally, their frame processing time $\Delta t_{\text{proc}^i_j}$ is:

$$\Delta t_{\text{proc}^i_j} = n_{\text{sim}} \times \Delta t^{(\cdot)}_{\text{sim1}}. \tag{22}$$

To assess this assumption, we have performed the following experiment: we have decoded different number of frames simultaneously in a single processor and evaluated the frame processing times in each case for the different type of frames. We have used JMVC 8.5 decoders [19] running in one of the cores of a general purpose PC. The results are shown in Table 2 shows the estimated frame processing times from $n_{\text{sim}} = 1$ to $n_{\text{sim}} = 4$, i.e., $\Delta t^{(\cdot)}_{\text{sim1}}$ to $\Delta t^{(\cdot)}_{\text{sim4}}$. The results show that our assumption is sufficiently good as the time devoted to decoding n_{sim} frames simultaneously is close to $n_{\text{sim}} \times \Delta t^{(\cdot)}_{\text{sim1}}$. In any case, we have assessed that the computational overhead that may occur when processing n_{sim} frames in n_{sim} threads of one processor do not incur in a processing time higher than $n_{\text{sim}} \times \Delta t^{(\cdot)}_{\text{sim1}}$.

To sum up, the estimation of $\Delta t_{\text{proc}^i_j}$, requires: (i) the computation of the number of frames that are simultaneously decoded at any time, (ii) the determination of the time intervals when parallel decoding of several frames occurs, and (iii) the policy on the assignment of simultaneously decoded frames to the available processors. In the next section we depict the iterative algorithm that we have employed to compute jointly $\Delta t_{\text{proc}^i_j}$ and the decoding chronogram.

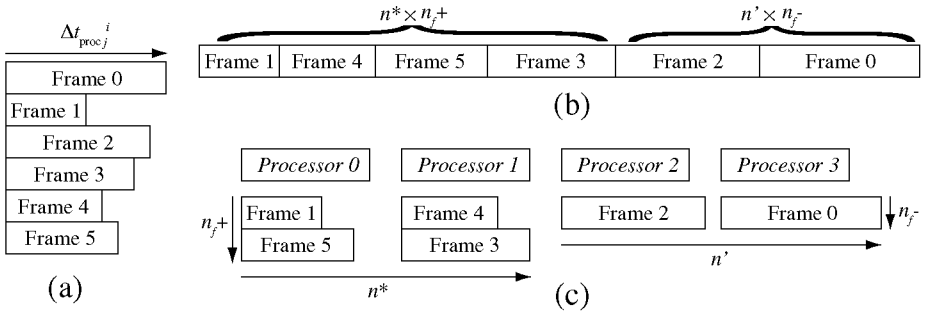


Fig. 5. Example on the assignment of frames to processors in the MPD model ($n_{\text{sim}} = 6$ and $K = 4$). The size of the frame bars represent $\Delta t_{\text{proc}^i_j}$ for each frame, at a given iteration. As a result of the assignment policy, frames 0 and 2 (frames with highest $\Delta t_{\text{proc}^i_j}$) are assigned to processors with lower computational load.

4. Iterative computation of the decoding delay in the Parallel MPD model

We have shown that in our decoder model, $\delta_{\text{dec}_j^i}$ can be computed with (20) and the DAG model. However, the values of $\Delta t_{\text{proc}_j^i}$ are not known a priori, as their values depend on the computational load conditions of the decoder that vary with time. To solve that, we use an iterative computation of the DAG. On each iteration, $\delta_{\text{dec}_j^i}$ and thus the decoding chronogram are computed using the DAG with the values of $\Delta t_{\text{proc}_j^i}$ obtained from the previous iteration. Then, the values of $\Delta t_{\text{proc}_j^i}$ are updated depending on the computational load conditions observed in the current decoding chronogram.

In this iterative algorithm, we make the following assumption: as the necessary sequential operations to decode a frame can be computed in different processors, we assume that at any time, the remaining operations of a decoding process can be assigned to any of the processors. For example, consider a decoder with two processors (P_0 and P_1) that are decoding three frames (x_0 in P_0 and x_1 and x_2 in P_1). If at a given time the decoding of x_0 ends, we can assign x_1 to P_0 and maintain x_2 in P_1.

4.1. Policy on the assignment of frames to processors

With the aim of developing a decoder model that has a fairly balanced processor usage, we implement an assignment policy on the MPD model that assigns frames with higher processing times to less loaded processors. This assignment policy is the following: consider that at given time and algorithm iteration, n_{sim} frames are being decoded simultaneously. If $n_{\text{sim}} \leqslant K$, each frame is assigned to one of the K processors and there is no simultaneous decoding within any processor. If $n_{\text{sim}} > K$, frames are assigned to the available processors by the following rules:

- The n_{sim} frames are distributed in K groups, in such a way that there will be groups with $n_f^- = \lfloor \frac{n_{\text{sim}}}{K} \rfloor$ frames and groups with $n_f^+ = \lceil \frac{n_{\text{sim}}}{K} \rceil$ frames, i.e., the maximum difference in number of frames among groups is one frame.
- The number of groups with n_f^+ frames, n^*, is:
$$n^* = n_{\text{sim}} \bmod K, \tag{23}$$
while the number of groups with n_f^- frames, n', is:
$$n' = K - n^*. \tag{24}$$
- Frames are distributed into the groups so that frames with higher $\Delta t_{\text{proc}_j^i}$ are assigned to groups with n_f^- frames. Thus, frames with higher $\Delta t_{\text{proc}_j^i}$ will be decoded in processors with lower computational load, limiting the extra decoding delay caused by parallel processing.
- Then, each of the K groups of frames is assigned to each one of the K processors.

An example of this assignment policy for $n_{\text{sim}} = 6$ and $K = 4$ is shown in Fig. 5. Fig. 5(a) shows, for a given time instant, a group of frames (frames 0 to 5) ready to be decoded with different values of $\Delta t_{\text{proc}_j^i}$. Fig. 5(b) shows those frames sorted by increasing processing time. It can be seen in Fig. 5(c) that the first $n_f^+ \times n^*$ are assigned in an alternate order to the first n^* processors and the last $n_f^- \times n'$ frames to the last n' processors.

4.2. Details of the iterative algorithm

This algorithm computes frame processing times and the decoding chronogram iteratively. On the initialization we assume that each frame is decoded in an exclusively dedicated processor. Then, on each iteration, the decoding chronogram is computed and we identify the time intervals in which the number of frames decoded simultaneously is higher than the number of processors. For those frames, we modify the frame processing times accordingly with the processor occupancy conditions. To update the frame processing times in the cases of simultaneous processing in one processor we follow the frame processing time model in Section 3.3. The flow diagram of this iterative algorithm is shown in Fig. 6.

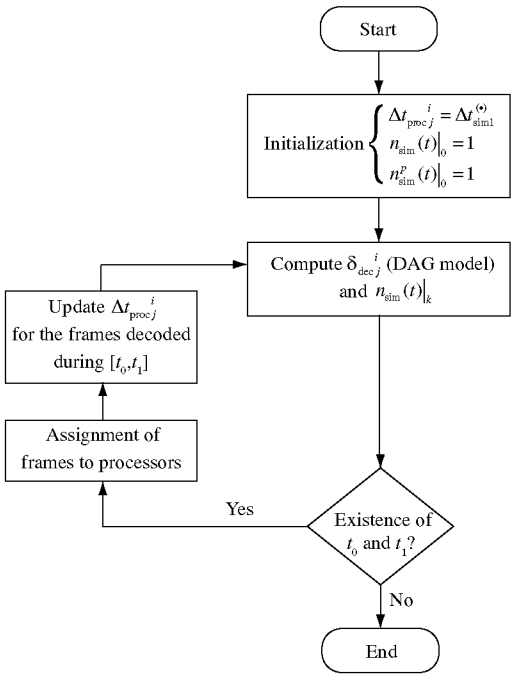


Fig. 6. Flow diagram of the iterative algorithm for the computation of decoding delay with the parallel MPD model.

- Iteration 0 (*Initialization of variables assuming that each frame is decoded in an exclusively dedicated processor.*)

1. The initial value of the frame processing time of each frame is set to $\Delta t_{\text{proc}_j^i}\big|_0 = \Delta t_{\text{sim1}}^{(\cdot)}, \forall i,j$.
2. The initial value of the number of frames that are decoded simultaneously over time is $n_{\text{sim}}(t)|_0 = 1$.
3. The initial value of the number of frames that are decoded simultaneously over time in processor p is $n_{\text{sim}}^p(t)\big|_0 = 1, \forall p$.

- Iteration k

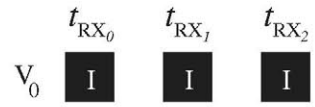
1. Computation of $\delta_{\text{dec}_j^i}, \forall i,j$ using the DAG model with $\Delta t_{\text{proc}_j^i}\big|_{k-1}$ and computation of $n_{\text{sim}}(t)|_k$ from the decoding chronogram results.
2. Comparison of $n_{\text{sim}}(t)|_k$ with $n_{\text{sim}}(t)|_{k-1}$. Define t_0 as the first instant for which $n_{\text{sim}}(t_0)|_k \neq n_{\text{sim}}(t_0)|_{k-1}$ and t_1 as the first instant after t_0 for which $n_{\text{sim}}(t_1)|_k \neq n_{\text{sim}}(t_0)|_k$. If these values do not exist finish the algorithm. If $n_{\text{sim}}(t_0)|_k = 0$, continue with 5. Otherwise, continue with 3.
3. Each frame which is being decoded during $\Delta t = [t_0, t_1]$, is assigned to one of the K processors by the assignment policy in Section 4.1, and $n_{\text{sim}}^p(t)\big|_k$ is updated $\forall t \in \Delta t$.
4. The processing time that is devoted to the decoding process of each frame in processor p during Δt according to the data from iteration $k-1$, $\widehat{C}_p(\Delta t)$, is:
$$\widehat{C}_p(\Delta t) = \frac{\Delta t}{n_{\text{sim}}^p(t_0)\big|_{k-1}} \tag{25}$$

while the processing time that is effectively devoted to the decoding process of each frame $C_p(\Delta t)$ is:

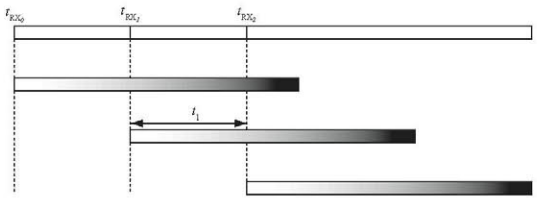
$$C_p(\Delta t) = \frac{\Delta t}{n^p_{\text{sim}}(t_0)\big|_k}. \tag{26}$$

The updated frame processing times of those frames are:

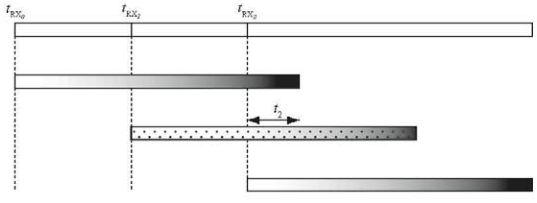
$$\Delta t_{\text{proc}^i_j}\Big|_k = \Delta t_{\text{proc}^i_j}\Big|_{k-1} + \Delta t'_{\text{proc}}, \tag{27}$$



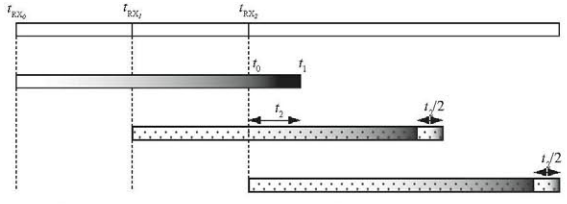
(a) Example prediction structure.



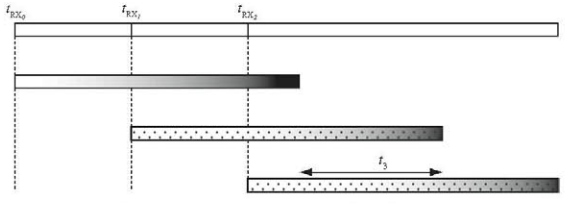
(b) Decoding chronogram. Iteration 1 (after initialization).



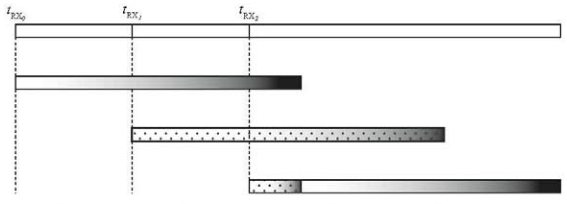
(c) Decoding chronogram. Iteration 2, after step 2.



(d) Decoding chronogram. Iteration 2, after step 4.



(e) Decoding chronogram Iteration 3.



(f) Decoding chronogram. Iteration 4 (Final iteration).

Fig. 7. Decoding chronograms in parallel MPD model. X-axis represents time and the different positions on the *y*-axis correspond to each of the frames of the prediction structure. t_{RX_i} are the reception times of the frames in the decoder. The horizontal bars represent the decoding time of each of the three frames. Different colors indicate the assignment of the frames to different processors, gradient: frame assigned to P_0, dots: frame assigned to P_1.

Table 3
Time parameter values for the MVC encoder [10].

Time parameter	Δt_{basic}	Δt_{ref}	Capture period
Value (ms)	20	10	40 (25 fps)

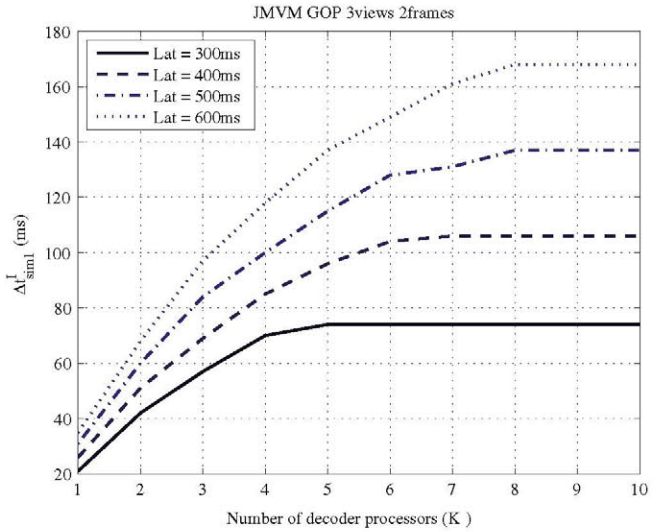


Fig. 8. Communication latency results. JMVM 3 views GOP2. Saturation on the value of Δt^I_{sim1}. The value of Δt^I_{sim1} cannot be increased beyond certain limit despite the number of decoder processors.

where $\Delta t'_{\text{proc}}$ is:

$$\Delta t'_{\text{proc}} = \widehat{C}_p(\Delta t) - C_p(\Delta t). \tag{28}$$

5. For $t > t_1, n_{\text{sim}}(t)|_k$ and $n^p_{\text{sim}}(t)|_k$ are reset to the initialization value for next iteration as follows: (*variables are reset to 1 for non-evaluated time periods.*)

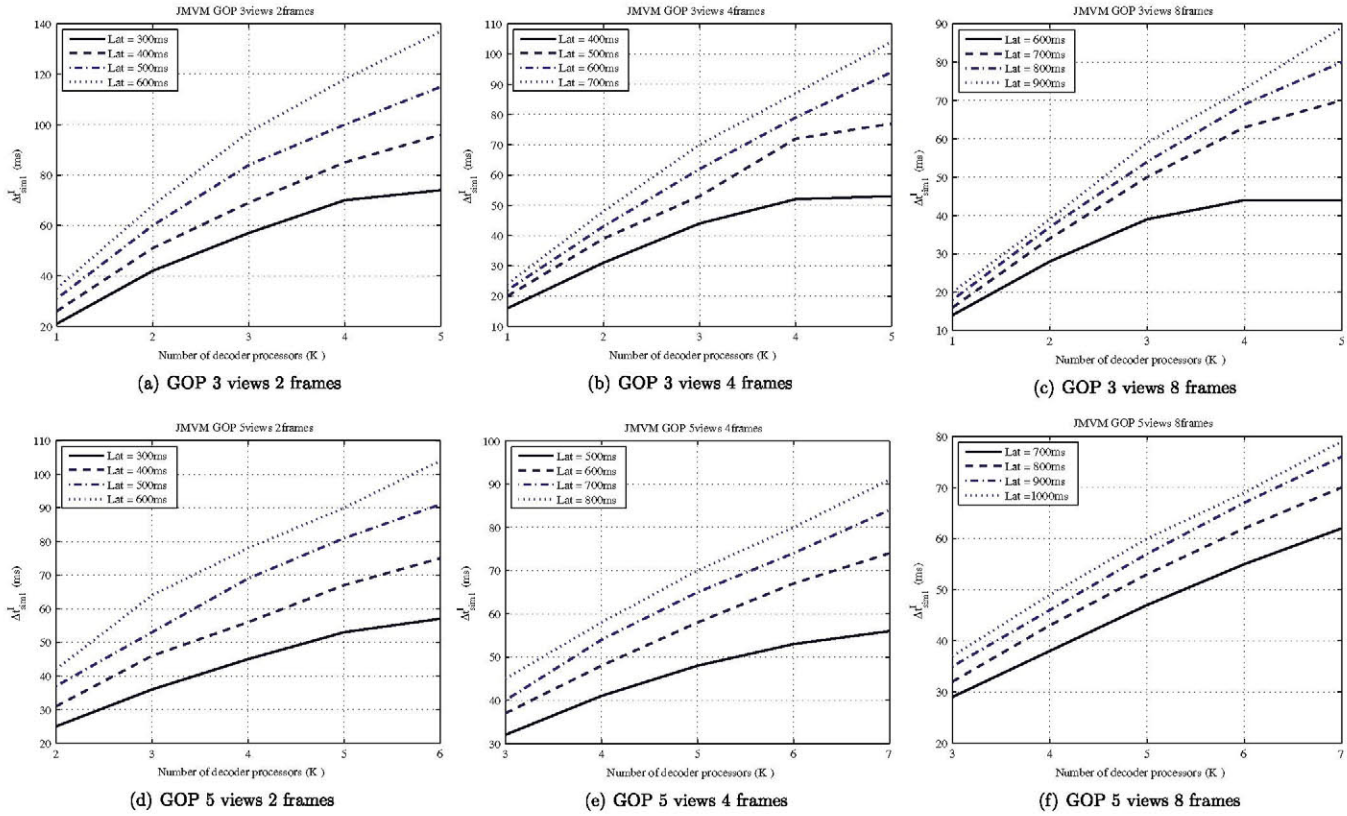
$$n_{\text{sim}}(t)|_k = n^p_{\text{sim}}(t)\big|_k = 1. \tag{29}$$

6. Go back to 1.

In order to illustrate the iterative algorithm, we show the example in Fig. 7. For simplicity reasons in this example, Fig. 7(a) shows a very simple GOP structure for one view with no prediction relationships. Fig. 7(b)–(e) show the decoding chronograms on several iterations of the algorithm for a decoder with $K = 2$ (processors P_0 and P_1). Fig. 7(b) shows the decoding chronogram as obtained in iteration 1 (after iteration 0) with $n_{\text{sim}}(t)|_0 = 1$ (initially, all the frames are assigned to P_0). By evaluation of the decoding chronogram, we find the interval Δt_1, in which $n_{\text{sim}}(\Delta t_1)|_1 = 2$. This means that during Δt_1 two frames are decoded simultaneously. As $K = 2$, the second frame is assigned to P_1 and processing times are not modified. After the second step of iteration 2 (Fig. 7(c)), the interval Δt_2 is found, for which $n_{\text{sim}}(\Delta t_2)|_2 = 3$. Thus, for frames x^0_0, x^0_1 and x^0_2, the frame processing times need to be updated. x^0_0 is assigned to P_0 and x^0_1, x^0_2 are assigned to P_1 and $\Delta t_{\text{proc}^0_1}$ and $\Delta t_{\text{proc}^0_2}$ are updated adding $\Delta t'_{\text{proc}} = \Delta t_2/2$, as shown in Fig. 7 (d) and fort $> t_1$, $n_{\text{sim}}(t)|_2 = 1$. On iteration 3 (Fig. 7 (e)), the interval Δt_3 is found, for which $n_{\text{sim}}(\Delta t_3)|_3 = 2$. Thus, the remaining decoding operations for x^0_2 are assigned to P_1 and frame processing times are not further modified. On iteration 4 (Fig. 7 (f)) there are no differences between $n_{\text{sim}}(t)|_3$ and $n_{\text{sim}}(t)|_4$ and the algorithm ends.

5. Experimental results

We have evaluated the communication latency of an MVC system, such as the one depicted in Fig. 2, for different multiview



(a) GOP 3 views 2 frames (b) GOP 3 views 4 frames (c) GOP 3 views 8 frames

(d) GOP 5 views 2 frames (e) GOP 5 views 4 frames (f) GOP 5 views 8 frames

Fig. 9. Combinations of number of processors and processor throughput Δt^{I}_{sim1} for a target communication latency value in a complete MVC system such as the one depicted in Fig. 2. Results for different JMVM prediction structures with hierarchical temporal prediction structure and IBP interview prediction [20].

prediction structures and MVC decoders with different processing capacities. We focus our analysis in the decoding delay, and the parameters of the MVC decoder within the Parallel MPD model that have an influence on it: the number of processors and processor throughput. Our approach on the tests is the following: given a prediction structure and a target value of the communication latency, we find possible combinations of those parameters that achieve the target latency value. To characterize the processor throughput, we use the processing time of a frame in an exclusively dedicated processor ($\Delta t^{(\cdot)}_{sim1}$) as the parameter under analysis. Note that $\Delta t^{(\cdot)}_{sim1}$ and the processor throughput (number of frames decoded per time unit) are inversely proportional.

We have performed this evaluation for different multiview GOP structures of the Joint Multiview Video Model (JMVM) with IBP prediction scheme for the interview prediction [20]. We have evaluated prediction structures with three and five views, and GOP sizes of two, four and eight frames. For all experiments, and as we are not focusing on the encoder delay, an MVC encoder with an unlimited processing capacity was assumed [10]. The frame processing time parameters for the encoder have been estimated in a general purpose PC: four-core processor working at 2.40 GHz, with 3.25 GB of RAM memory. Also, for simplicity in our simulations the values of the transmission delay $\delta_{RX^i_j}$ are not considered. The encoder time parameter values are shown in Table 3. For the frame processing time model in the decoder, we have used the following parameters: $\alpha_P = 0.6, \alpha_B = 0.8$.

Fig. 8 shows, for a GOP of three views and size of two frames, the evolution of the maximum value of Δt^{I}_{sim1} that guarantees a communication latency below a target value, with different numbers of processors. Results are shown for several target communication latency values. For example, considering the graph of $Lat = 500$ ms and a decoder with two processors, Δt^{I}_{sim1} must be under 60 ms to obtain a communication latency below 500 ms. Alternatively, if $\Delta t^{I}_{sim1} = 100$ ms, the decoder needs at least four processors to obtain a communication latency below 500 ms. It can be seen that each of the graphs reaches a saturation value of Δt^{I}_{sim1}, $\Delta t^{I}_{sim1,max}$. This result indicates that there exists a limit to the frame processing time to guarantee the target latency value despite the number of processors. This value is obtained when the number of processors is equal to the maximum number of frames that have to be decoded in parallel at any time.

Fig. 9 shows the same type of results for different JMVM prediction structures. With these results, we prove that the proposed framework allows us to solve design problems on MVC decoders such as: given a target communication latency, a prediction structure and a certain processor throughput, we can find the minimum number of processors to achieve a communication latency under that target value. Alternatively, given a number of processors we can compute the maximum value of frame processing time that guarantees the target communication latency.

6. Conclusions

We have presented a framework for the systematic analysis of the decoding delay of multiview decoders. We have shown that in real-time applications, an accurate estimation of the decoding delay is an essential factor to achieve a minimum communication latency.

Thus, the proposed framework completes the analysis of the contribution of the MVC codec to that communication latency. The delay on the decoder depends on: (i) the multiview prediction structure and (ii) the hardware architecture of the decoder. We have considered a multi-processor platform with multi-threading capabilities, whose main characteristics are captured in the Parallel

MPD model. We have shown that the contribution of the multiview prediction structure to the decoding delay can be computed using graph theory, given that the frame processing times are known. As in the Parallel MPD model the frame processing times depend on the computational load of the processors in the decoder, we have provided an iterative algorithm to compute jointly frame processing times and the decoding delay in such a decoder platform.

Finally, we have shown that this framework can be applied to design decoders with the aim of minimizing the communication latency. It provides a tool for an efficient design of characteristics of the decoder that have an influence on the decoding delay performance, such as the number of processors or the processor throughput. We have shown that given a prediction structure and a given processor throughput we are able to find the minimum number of decoder processors to achieve a target communication latency value. Alternatively, given the number of processors we find the minimum processor throughput to achieve that target value.

Acknowledgements

This work has been partially supported by the Ministerio de Economía y Competitividad of the Spanish Government under project TEC2010-20412 (Enhanced 3DTV). Also, P. Carballeira wishes to thank the Comunidad de Madrid for a personal research grant.

References

[1] A. Smolic, K. Müller, P. Merkle, C. Fehn, P. Kauff, P. Eisert, T. Wiegand, 3D video and free viewpoint video – technologies, applications and MPEG standards, in: Proceedings of IEEE International Conference on Multimedia and Expo, 2006, pp. 2161–2164.

[2] Y. Morvan, D. Farin, P. De With, System architecture for free-viewpoint video and 3D-TV, IEEE Transactions on Consumer Electronics 54 (2008) 925–932.

[3] S.B. Kang, R. Szeliski, P. Anandan, The geometry-image representation tradeoff for rendering, in: Proceedings of IEEE International Conference on Image Processing, vol. 2, 2000, pp. 13–16.

[4] P. Merkle, A. Smolic, K. Müller, T. Wiegand, Multi-view video plus depth representation and coding, in: Proceedings of IEEE International Conference on Image Processing, 2007, pp. 201–204.

[5] A. Vetro, P. Pandit, H. Kimata, A. Smolic, Y. Wang, Joint Draft 8.0 on Multiview Video Coding, Doc. JVT-AB204 Hannover, Germany, July, 2008.

[6] ITU-T Rec. H.264 & ISO/IEC 14496-10, AVC Advanced Video Coding for Generic Audiovisual Services, 2005.

[7] A. Vetro, T. Wiegand, G. Sullivan, Overview of the stereo and multiview video coding extensions of the H.264/MPEG-4 AVC standard, Proceedings of the IEEE 99 (2011) 626–642.

[8] Y. Liu, Q. Huang, S. Ma, D. Zhao, W. Gao, RD-optimized interactive streaming of multiview video with multiple encodings, Journal of Visual Communication and Image Representation 21 (2010) 523–532.

[9] G. Karlsson, Asynchronous transfer of video, IEEE Communications Magazine 34 (1996) 118–126.

[10] P. Carballeira, J. Cabrera, A. Ortega, F. Jaureguizar, N. Garcia, A framework for the analysis and optimization of encoding latency for multiview video, IEEE Journal of Selected Topics in Signal Processing 6 (2012) 583–596.

[11] Y. Yang, G. Jiang, M. Yu, D. Zhu, Parallel process of hyper-space-based multiview video compression, in: Proceedings of IEEE International Conference on Image Processing, 2006, pp. 521–524.

[12] Y. Pang, L. Sun, J. Wen, F. Zhang, W. Hu, W. Feng, S. Yang, A framework for heuristic scheduling for parallel processing on multicore architecture: a case study with multiview video coding, IEEE Transactions on Circuits and Systems for Video Technology 19 (2009) 1658–1666.

[13] ITU-T Rec. G.1050, Network model for evaluating multimedia transmission performance over internet protocol, 2007.

[14] K. Thulasiraman, M.N.S. Swamy, Graphs: Theory and Algorithms., Wiley, 1992.

[15] A. Vetro, M. McGuire, W. Matusik, A. Behrens, J. Lee, H. Pfister, Multiview Video Test Sequences from MERL, MPEG Contribution M12077, Busan, Korea, 2005.

[16] Y. Ho, E. Lee, C. Lee, Multiview Video Test Sequence and Camera Parameters, MPEG contribution M15419, Archamps, France, 2008.

[17] M. Tanimoto, T. Fujii, M.P. Tehrani, M. Wildeboer, N. Fukushima, H. Furihata, Moving Multiview Camera Test Sequences for MPEG-FTV, MPEG contribution M16922, Xi'an, China, 2009.

[18] P. Carballeira, J. Cabrera, A. Ortega, F. Jaureguizar, N. García, Comparative latency analysis for arbitrary multiview video coding prediction structures, in: Proceedings of IS&T/SPIE Visual Communications and Image Processing, 2009, vol. 7257, pp. 72570L–1–12.

[19] Joint Video Team, JMVC Reference Software 8.5, <www.garcon.ient.rwthaachen.de>, 2011.

[20] A. Vetro, P. Pandit, H. Kimata, A. Smolic, Y. Wang, Joint multiview video model (JMVM) 8.0, output doc. N9762, Archamps, France, April 2008.