The Journal of Supercomputing manuscript No.
(will be inserted by the editor)

A highly scalable parallel encoder version of the emergent JEM video encoder

H. Migallón · M. Martínez-Rach · O. López-Granado · V. Galiano · M.P. Malumbres · Glenn Van Wallendael

Received: date / Accepted: date

Abstract In 2016, 73% of total internet traffic came from video transmission and this percentage is expected to reach 82% by 2021. These figures show the importance of using video compression standards that maximize video quality while minimizing the necessary bandwidth. In 2013, the HEVC standard was released accounting for an approximate 50% bit-rate saving compared to H.264/AVC while maintaining the same reconstruction quality. To address increases in video IP traffic, a new generation of video coding techniques is required that achieve higher compression rates. Compression improvements are being implemented in a software package known as the Joint Exploration Test Model. In this work, we present two parallel JEM model solutions specifically designed for distributed memory platforms for both All Intra and Random Access coding modes. The proposed parallel algorithms achieved high levels

This research was supported by the Spanish Ministry of Economy and Competitiveness under Grant TIN2015-66972-C5-4-R, co-financed by FEDER funds.(MINECO/FEDER/UE)

H. Migallón
Physics and Computer Architecture Dept. Miguel Hernández University, 03202 Elche, Spain.
Tel.: +34-966658390
Fax: +34-966658814
E-mail: hmigallon@umh.es

M. Martínez-Rach
Physics and Computer Architecture Dept. Miguel Hernández University, 03202 Elche, Spain.

O. López-Granado
Physics and Computer Architecture Dept. Miguel Hernández University, 03202 Elche, Spain.

V. Galiano
Physics and Computer Architecture Dept. Miguel Hernández University, 03202 Elche, Spain.

M.P. Malumbres
Physics and Computer Architecture Dept. Miguel Hernández University, 03202 Elche, Spain.

Glenn Van Wallendael
Multimedia Lab, Ghent University, Belgium.

of efficiency, in particular for the All Intra mode. They also showed great scalability.

Keywords JEM, Video coding, Parallel algorithms, Multicore, Performance

1 Introduction

High Efficiency Video Coding (HEVC) [17] was developed in 2013 by the Joint Collaborative Team on Video Coding (JCT-VC) to replace the H.264/-Advanced Video Coding (AVC) standard [10]. The HEVC standard achieves a bit rate saving of almost 50% compared to the previous video coding standard, but at a cost of high computational complexity during the encoding [14].

According to the 'The Zettabyte Era: Trends and Analysis' Cisco report [6], IP video traffic represented 73% of total IP traffic in 2016, and is expected to reach 82% by 2021. This means that each second, a million minutes of video content travels through the network. The report also predicts a constant increase in novel services such as Video-on-Demand (VoD), Live internet video, as well as Virtual Reality (VR) and Augmented Reality (AR). Thus, VoD traffic will double by 2021 mainly due to the increasing number of consumers and greater video resolution (4K and 8K). There is also intense interest in live video streaming of sports events, such as soccer or American football where major teams are installing several 4K and 5K resolution cameras (more than 20) around the stadiums to transmit a 360^o view of the matches.

To address this increase in video IP traffic, a new generation of video coding techniques is required to achieve higher compression rates than the ones obtained by previous standards. Since the release of HEVC, both the ITU-T Video Coding Expert Group (VCEG) and the ISO/IEC Moving Picture Expert Group (MPEG) have been studying new video coding technologies with a compression capability that significantly exceeds that of HEVC, with the aim of developing a new video coding standard. To do so, a framework of collaboration has been created called the Joint Video Exploration Team (JVET).

These compression enhancements are being implemented by the JVET in a software package known as the Joint Exploration test Model (JEM) [5]. Its main purpose is to investigate gains from additional coding tools in the video coding layer. It should be noted that the main purpose of JEM is not to establish a new standard, but to study which new modifications beyond HEVC will be worthy of interest in terms of compression performance. The main goal of this possible new coding standard is to achieve bit rate savings between 25% to 30% compared to HEVC [2,15].

Experimental results using the All Intra (AI) configuration [12] show that the new model (JEM 3.0) achieves an 18% bit rate reduction, but at the expense of an extremely high increase in computational complexity (60x) with respect to HEVC. On the other hand, with the Random Access (RA) configuration, JEM obtains on average a bit rate reduction of 26% with a computational complexity increment of (11x).

This increase in high complexity makes it necessary to introduce acceleration techniques and leverage hardware architectures to reduce encoding time. As JEM is still not fully deployed, only a few articles have been published on the subject, and most of them focus on rate distortion (R/D) comparisons between JEM, HEVC and AV1 codecs [8]. Recently, a proposal [7] was put forward to accelerate the Motion Estimation (ME) stage. In this latter paper, the authors proposed a pre-analysis algorithm designed to extract motion information from a frame, later used in the ME module to speed up the encoder. They thus showed that around 27% of the reference frames could be skipped and that more than 62% of time was saved on the integer ME operation with a negligible impact of 0.11% on the Bjøntegaard delta rate (BD-rate).

In this paper, we present a JEM model parallel encoder specifically designed for distributed memory platforms for both All Intra (AI) and Random Access (RA) coding modes, especially suited for video editing and video broadcast applications respectively. We performed several experimental tests to illustrate the behaviour of the parallel versions in terms of efficiency and scalability.

The rest of this paper is organized as follows: in Section 2, a brief description of the coding tools introduced in the new JEM video coding standard is presented; in Section 3, a JEM video standard is compared with respect to the previous HEVC standard; in Section 4, the parallel algorithms for RA and AI coding modes of the JEM video coding standard are described; in Section 5 we present test results of the parallel algorithms. We draw conclusions in Section 6.

2 Description of the algorithm of the Joint Exploration Test Model (JEM)

As the JEM codec is based on the HEVC reference software, called HM, the overall architecture of the JEM codec is quite similar to that of the HEVC HM codec. Both codecs use closed-loop prediction with motion compensation from previously decoded reference frames, or intra prediction from previously decoded areas of the current frame. The improvements to the coding layer by JEM were related to the following aspects:

– **Picture partitioning**. In HEVC, each video sequence frame is divided into a set of non-overlapping blocks[17], where each block corresponds to a quad tree structure called CTU [18]. These structures (CTUs) can in turn be partitioned into coding units (CU), prediction units (PU) and transform units (TU). PUs store prediction information such as Motion Vectors (MVs) and can range from 64x64 to 8x8 using either symmetrical or asymmetrical sizes. For this purpose, HEVC defines eight possible partitions for each CU size: 2Nx2N, 2NxN, Nx2N, NxN, 2NxnU, 2NxnD, nLx2N and nRx2N.
  In the JEM, the highest coding level is called the Coding Tree Unit (CTU) as in HEVC. One of the main changes in the JEM is that block splitting

below the CTU level can be performed either by quad and/or binary split steps. This partition method is called quad tree plus binary tree (QTBT). This structure gives more flexibility to block partition shapes to better match the local characteristics of the video sequence. The organization in Coding Units (CU), Prediction Units (PU) and Transform Units (TU) is thus no longer needed [11]. In QTBT, CUs can have either square or rectangle shapes like PUs in HEVC. Each CTU, which can measure up to $256x256$ pixels, is first partitioned by a quad-tree structure in squared CUs as in the HEVC. Leaf nodes can then be further partitioned by a binary tree structure. In this binary tree, each CU can be split into two horizontal or vertical halves.

- **Spatial prediction**. To detect finer edge directions, the directional intra modes are extended from 33, as defined in the HEVC, to 65 modes. These dense directional intra prediction modes apply to all block sizes from both luma and chroma intra predictions, while in the HEVC only luma intra prediction is computed. To adapt the greater number of directional intra-modes, an intra-mode coding acceleration method is defined with 6 Most Probable Modes instead of the 3 used in the HEVC.
- **Temporal prediction**. Regarding inter-prediction, with QTBT, each CU can have one set of motion information for each prediction direction at the most. Despite this, two sub-CU level motion vector prediction methods are studied by splitting a large CU into sub-CUs and deriving motion information for all the sub-CUs of the large CU. Temporal motion vector prediction is also supplemented by more advanced mechanisms and by increasing the resolution of the reference vectors. Furthermore, Overlapped Block Motion Compensation (OBMC) and Local Illumination Compensation (LIC) are used.
- **Transform improvements**. In JEM, in addition to the Discrete Cosine Transform (DCT)-II and the $4x4$ Discrete Sine Transform (DST)-VII used in the HEVC for transform coding, an Adaptive Multiple Transform (AMT) scheme has been chosen to encode the inter and intra residuals. It uses different DCT and DST families from the ones used in HEVC. Furthermore, an intra mode dependent non-separable secondary transform (NSST) is defined. JEM also implements a specific Signal Dependent Transform (SDT), which determines the best transform basis from already decoded neighbouring samples.
- **Loop filter**. In JEM, an Adaptive Loop Filter (ALF) with block-based filter adaption is applied. For the luma component, according to the direction and activity of local textures, one filter is selected among 25 filters for each 2×2 block [4]. ALF aims to reduce visible artifacts such as ringing and blurring by reducing the mean absolute error between the original image and the reconstructed image. Further, a bilateral filter is operated directly following the inverse transform, denoising flat areas in combination with edge preservation.
- **Entropy coding**. In the HEVC, the entropy coder used is Context-based Adaptive Binary Arithmetic Coding (CABAC). JEM uses an enhanced

version of CABAC with modified context modelling for transform coefficients, a multi-hypothesis probability estimation with context-dependent updating speed, and an adaptive initialization of models.

3 Comparative analysis between HEVC and JEM

In this section, we present a comparison in terms of R/D (Rate/Distortion) and encoding time between HEVC and JEM encoding standards using both the AI and RA coding modes in their sequential versions. Four video sequences described in Table 1 with different resolutions were used in our study. The reference encoder software was JEM 7.0 [5] and for the previous HEVC standard, the HM 16.3 [9] was used.

Table 1 Test video sequences.

Video sequence	Acronym	Resolution	Frame rate (Hz)	Num. of frames	Video time (s)
People on Street	PEOPON	2560x1600	30	150	5
Park Scene	PKSCNE	1920x1080	24	240	10
Four People	FOURPE	1280x720	60	600	10
Party Scene	PARTYS	832x480	50	500	10

Table 2 shows the BD-rate, which represents the percentage of bit rate variation between two sequences with the same objective quality [3]. Therefore, a negative value implies that the proposal improves the coding efficiency of the baseline encoder. As illustrated, the JEM encoder outperformed the HEVC encoder. The maximum gain in the AI coding mode was 18.88% and it was obtained for the highest resolution video sequence. The average BD-rate gain for the AI coding mode was 13.9% for the tested video sequences. Looking at the results for the RA coding mode, the JEM encoder obtained better results than the HEVC. The maximum gain in the RA coding mode was 31.6% for the 'Four People' video sequence, the average gain for all tested video sequences being 28.82%.

Table 2 also shows the average encoding time increase required by the JEM encoder with respect to the HEVC one for both AI and RA coding modes. As shown, the JEM encoder required, on average, 41 times the total time required by HEVC software to encode the tested sequences in the AI coding mode, and 7 times the time in the RA coding mode. To illustrate the huge complexity increase of the JEM encoder, to encode 150 frames of the 'People on Street' video sequence with a QP value of 22 in AI coding mode, the JEM encoder required 95.5 hours, whereas the HEVC encoder only needed 1.71 hours.

Based on previous results, big efforts must be undertaken to speed-up the total encoding time of the new JEM video coding model.

In this paper, we present two parallel versions of the JEM encoder specifically designed for digital cinema (AI) and video on demand (RA). In the

Table 2 Comparison between HEVC and JEM encoders in terms of R/D and complexity

	AI mode		RA mode	
Video sequence	BD-Rate (%)	Increase in encoding time	BD-Rate (%)	Increase in encoding time
PEOPON	-18.88%	44.53x	-31.00%	8.71x
PKSCNE	-13.31%	44.99x	-26.10%	6.51x
FOURPE	-17.60%	34.58x	-31.60%	4.46x
PARTYS	-5.84%	44.99x	-26.60%	8.54x

case of the RA coding mode parallel versions, we used Instantaneous Decoder Refresh (IDR) intra pictures instead of typical Clean Random Access (CRA) intra pictures used in the sequential version. As explained in [16], an IDR picture clears the contents of the reference picture buffer and thus all the following pictures can be decoded without references to any frame preceding the IDR picture. Contrary to IDR pictures, and to improve coding efficiency, CRA pictures allow pictures that follow the CRA picture in decoding order but precede them in output order to use pictures decoded before the reference CRA picture and still allow similar clean random access functionality as an IDR picture.

To show the effect of using IDR pictures instead of CRA pictures in R/D, Table 3 illustrates the BD-rate increase when using IDR pictures instead of CRA pictures at different intra periods. As expected, the use of IDR pictures produced an increase in the final bit rate, especially for the lowest intra period. Although a small increment was introduced by the IDR pictures, they allowed us to break frame dependencies at each intra period, making the distribution of Group of Pictures (GOPs) possible between different computing nodes in the parallel versions.

Table 3 R/D comparison between IDR and CRA intra frames at different Intra periods in the JEM encoder

Video sequence	IntraPeriod 32 BD-Rate (%)	IntraPeriod 48 BD-Rate (%)	IntraPeriod 64 BD-Rate (%)
PEOPON	5.5%	4.1%	2.8%
PKSCNE	13.3%	7.6%	5.8%
FOURPE	11.8%	8.0%	6.2%
PARTYS	10.4%	6.9%	4.8%

4 Parallel Algorithms

We designed and developed parallel algorithms of the JEM video coding model based on a GOP structure for both RA and AI coding modes, namely PJEM-

GOP-RA and PJEM-GOP-AI respectively. The characteristics of a GOP depends on the selected coding mode. A GOP consists of a single I-frame when the AI coding mode is used, whereas for the RA coding mode, a GOP consists of 16 B-frames, i.e. frames that use both spatial and temporal redundancy in the encoding procedure. The RA coding mode periodically inserts an I-frame, i.e. the coding mode is changed from B-frame to I-frame. This period is known as 'Intra Period' and is specified as an integer number of frames. The 'Intra Period' has two restrictions: first, the value has to be a multiple of the GOP size, and second, a minimum value of at least two GOPs (in number of frames) is required when IDR is used as 'Decoding Refresh Type'. As mentioned above, an I-frame of IDR type implies that none of the consecutive frames will use temporal redundancy with frames prior to that I-frame.

As the GOP structure is quite different for each mode, we developed specific algorithms. Both algorithms consisted in one manager process and a set of encoding processes. The manager process distributes the workload among the encoding processes as they become idle, thus balancing the workload. The parallel algorithms were developed in order to use hybrid memory computing platforms, therefore the computing platform was managed through MPI (Message Passing Interface).

The algorithm developed for the RA coding mode is shown in Algorithms 1 and 2, which respectively describe the parallel encoding procedure and the manager procedure. MPI processes are identified with a natural number between zero and the total number of processes minus one, and the manager process is always the last MPI process. Moreover, both the encoding process 0 and the manager process are mapped on a single computing node, and, if necessary, in the same core. Algorithm 1 shows the parallel encoding procedure for the PJEM-GOP-RA algorithm. The encoding processes receive $DecodingRefreshType$ and $IntraPeriod$ parameters from the manager process, because both parameters depend on the block structure and they are computed by the manager process. Next, each encoding process requests the first GOP block (group of consecutive GOPs) to be encoded in the manager process. In lines 9–11 and 22–24 a GOP block is encoded, i.e. a group of $NumGOPsPerBlock * GOPSize$ frames are encoded. The first frame of this group of frames must be an IDR I-frame, since the previous encoded frames, if any, are almost certainly not the previous frames at temporal level. In fact, the previous frames at temporal level have most likely been or will be encoded by another process. Therefore, even the initial IDR I-frame allows us to break temporal dependencies.

In Algorithm 2, the first frame of a block of GOPs is guaranteed to be an IDR-frame, setting the $DecodingRefreshType$ parameter equal to 2 (all I-frames will be of IDR type), while also setting the $IntraPeriod$ parameter equal to the size in frames of the GOP block. These two parameters and the number of GOPs in one block must be transmitted to all encoding processes. Algorithm 2 is divided into three main parts, the first one distributes the initial workload, the second one stores the bitstreams received and distributes the remaining workload, and the third one sends the signal to all encoding processes

Algorithm 1 PJEM-GOP-RA: Parallel algorithm for encoding processes.

```
1: Define configuration parameters
2: Encoding processes:
3: {
4:    Receive (from broadcasting) DecodingRefreshType, IntraPeriod and
      NumGOPsPerBlock parameters
5:    GOPBlockRead = 0
6:    Send MPI message requesting new work
7:    Receive MPI message with BlockGOPToEncode
8:    Move file pointer to frame BlockGOPToEncode * NumGOPsPerBlock * GOPSize
9:    for i = 1 to NumGOPsPerBlock do
10:      Encode GOP
11:   end for
12:   while true do
13:      Send MPI message with bitstream size of NumGOPsPerBlock GOPs encoded
14:      Send MPI message with bitstream data and requesting new work
15:      Receive MPI message with BlockGOPToEncode and EndSignal
16:      if EndSignal == True then
17:         Break while
18:      else
19:         Disk file displacement (in number of frames):
20:          (BlockGOPToEncode − GOPBlockRead) * NumGOPsPerBlock * GOPSize
21:         Move file pointer to new initial frame of the new GOP
22:         for i = 1 to NumGOPsPerBlock do
23:            Encode GOP
24:         end for
25:         GOPBlockRead = BlockGOPToEncode
26:      end if
27:   end while
28: }
```

to stop the encoding procedure. The manager process must compose the final bitstream correctly. This reordering procedure is explained in Algorithm 4.

Algorithms 3 and 4 describe the parallel algorithm developed for the AI coding mode, for the parallel encoding processes and the manager process respectively. As shown in Algorithm 3, the encoding processes start the encoding procedure immediately, without requesting a workload to the manager process. The initial frame encoded by each MPI process corresponds to the frame in the same order as the MPI rank of the encoding process. When the frame encoding has ended, the process will send the bitstream and will request the order of the new frame to be encoded. If the manager process replies 0, the encoding process ends the encoding procedure and waits for the MPI execution to terminate.

Algorithm 4 shows the management procedure to distribute the encoding work and compose the final bitstream. Worthy of note, this algorithm, like Algorithm 2, includes an intrinsic work balancing procedure, because the encoding work is assigned according to the demand. Therefore, since the order of reception of the encoded frames (bitstreams) may not be the right order, the manager process has to reorder the received bitstreams. Algorithm 4 is divided into two main parts: the first distributes the workload and stores the

Algorithm 2 PJEM-GOP-RA: Manager process algorithm.

```
Set DecodingRefreshType = 2
Read NumGOPsPerBlock
Set IntraPeriod = NumGOPsPerBlock * GOPSize
Manager process:
{
    Broadcast DecodingRefreshType, IntraPeriod and NumGOPsPerBlock parameters
    Obtain the number of TotalGOPBlocks
    BlockGOPToEncode = 0
    for i = 1 to NumMPIProcesses − 1 do
        Receive MPI message requesting new work
        Send MPI message with BlockGOPToEncode
        BlockGOPToEncode + +
    end for
    for i = BlockGOPToEncode to TotalGOPBlocks do
        Receive MPI messages encoded data
        Send MPI message with BlockGOPToEncode
        Store encoded data in final bitstream or in temporal memory
        Take data from temporal memory, if any, to be added to the final bitstream
        BlockGOPToEncode + +
    end for
    for i = 1 to NumMPIProcesses − 1 do
        Receive MPI message with bitstream size
        Receive MPI messages encoded data
        Store encoded data in final bitstream or in temporal memory
        Take data from temporal memory, if any, to be added to the final bitstream
        Send MPI message with EndSignal = True
    end for
}
```

Algorithm 3 PJEM-GOP-AI: Parallel algorithm for encoding processes.

```
Define encoding configuration parameters
Encoding processes:
{
    FrameRead = MPIRank
    Move file pointer to frame FrameRead
    Read frame from disk file
    Encode as I-frame
    while true do
        Send MPI message with bitstream size
        Send MPI message with bitstream data and requesting new frame to encode
        Receive MPI message with FrameToEncode
        if FrameToEncode == 0 (No remaining work) then
            Break while
        else
            Move file pointer to frame FrameToEncode
            Read frame from disk
            FrameRead = FrameToEncode
            Encode as I-frame
        end if
    end while
}
```

bitstreams received, and the second sends the 'end of work' signal to all encoding processes.

Algorithm 4 PJEM-GOP-AI: Manager process algorithm.

```
1: Manager process:
2: {
3:    Obtain the number of TotalFramesToEncode
4:    FrameToEncode = NumMPIProcesses
5:    FrameToStore = 0
6:    for i = 0 to NumMPIProcesses − 2 do
7:       FrameEncodedInProcess[i] = i
8:    end for
9:    for i = NumMPIProcess to TotalFramesToEncode do
10:      Receive MPI message with bitstream size
11:      MPIProcessReceiving = MPIProcessOfPreviousMessage
12:      Receive MPI message with bitstream data from process MPIProcessReceiving
13:      if FrameEncodedInProcess[MPIProcessReceiving] == FrameToStore then
14:         FrameToStore + +
15:         Store bitstream received in final bitstream
16:         repeat
17:            TestFlag = False
18:            for all bitstreams stored in temporal memory do
19:               if NumFrameOfBitstream == FrameToStore then
20:                  Store bitstream received in final bitstream
21:                  Remove bitstream from temporal memory
22:                  TestFlag = True
23:                  FrameToStore + +
24:                  Break ForEach
25:               end if
26:            end for
27:         until TestFlag == True
28:      else
29:         Store bitstream received in temporal memory
30:      end if
31:      Send MPI message with FrameToEncode
32:      FrameEncodedInProcess[MPIProcessReceiving] == FrameToEncode
33:      FrameToEncode + +
34:   end for
35:   for i = 1 to NumMPIProcesses − 1 do
36:      Receive MPI message with bitstream size
37:      Receive MPI message with bitstream data
38:      Send MPI message with FrameToEncode = 0
39:   end for
40: }
```

5 Numerical experiments

In this section we analyze both the parallel behaviour and the encoding performance of the parallel proposals. The reference encoder software is JEM 7.0 [5], and to perform the tests, the GCC v.4.8.5 compiler [1] and MPI v2.2 [13] were used. The parallel platform used was composed of HP Proliant SL390 G7

computing nodes, where each node was equipped with two Intel Xeon X5660 processors. Each X5660 included six processing cores at 2.8 GHz. Moreover, a QDR Infiniband was used as a communication network. In the experiments performed, only one MPI process was mapped in each computing node, i.e. a strict distributed memory platform was used.

As mentioned above, $DecodingRefreshType$ and $IntraPeriod$ parameters must be set according to the parallel algorithm for the RA coding mode. The first parameter sets all I-frames as IDR frames, and the second one should be increased depending on the GOP block size ($NumGOPsPerBlock$ parameter). Table 4 compares the parallel algorithms in terms of R/D (Rate/Distortion). The reference algorithm in Table 4 is the sequential algorithm with I-frames of CRA type instead of IDR, and an $IntraPeriod$ parameter equal to 32. As illustrated in TableTable 4, using the minimum GOP block size (equal to 2), the average BD-rate increase is equal to 10.2%, but with a lower complexity. When the GOP block size is increased, the BD-rate increment is reduced whereas computational complexity becomes similar.

Table 4 PJEM-GOP-RA: Comparison between parallel and sequential JEM encoders in terms of (R/D) and computational complexity increment.

Video sequence		Number of GOPs per block 2	3	4
PEOPON	BD-Rate(%)	11,8%	-8,1%	-18,1%
PKSCNE		13,3%	0,2%	-3,1%
FOURPE		10,4%	-0,9%	-6,2%
PARTYS		5,5%	2,6%	0,0%
PEOPON	Inc. Complexity(%)	-3.3%	-3.0%	-3.5%
PKSCNE		-5.0%	-2.9%	-0.6%
FOURPE		-5.7%	-2.7%	-1.0%
PARTYS		-3.1%	-0.5%	-0.8%

Before addressing the results on the efficiency of the PJEM-GOP-RA parallel algorithm, Table 5 illustrates the maximum number of processes allowed depending on the $NumGOPsPerBlock$ parameter (2, 3 or 4)) which sets the value of the $Intraperiod$ parameter (32, 48 or 64). Therefore, the parallel efficiency analysis will be performed using a lower number of processes. Worthy of note, if longer sequences were used, the encoding performance analysis will still be valid for a larger number of processes.

Tabletab:EfficiencyRA shows the average efficiency for all QP values used (22, 27, 32, 37), obtained by the PJEM-GOP-RA parallel algorithm. The efficiency is on average 88%. As shown, when all computational work is divided into a small number of blocks, the computational load can become unbalanced and the parallel algorithm is not able to balance the workload. However, parallel efficiency increases for higher resolution video sequences.

Table 5 PJEM-GOP-RA: Maximum number of processes.

Video sequence (Number of frames)	Number of GOPs per block (Intra period)		
	2 (32)	3 (48)	4 (64)
PEOPON (150)	4 pr.	3 pr.	2 pr.
PKSCNE (240)	7 pr.	5 pr.	3 pr.
FOURPE (600)	18 pr.	12 pr.	9 pr.
PARTYS (500)	15 pr.	10 pr.	7 pr.

Table 6 PJEM-GOP-RA: Parallel efficiency.

Video sequence	Num. GOPs per block	Intra period	Number of processes 2 pr.	3pr.	4pr.
PEOPON	2	32	88%	97%	95%
	3	48	99%	99%	N/A
	4	64	95%	N/A	N/A
PKSCNE	2	32	77%	74%	77%
	3	48	87%	93%	92%
	4	64	77%	93%	N/A
FOURPE	2	32	91%	84%	83%
	3	48	90%	90%	87%
	4	64	83%	91%	90%
PARTYS	2	32	87%	83%	80%
	3	48	88%	91%	87%
	4	64	80%	89%	77%

Table 7 shows parallel efficiencies when a 'Four People' sequence is encoded using 6, 8 and 10 processes. As can be seen, the parallel efficiency of the PJEM-GOP-RA parallel algorithm increased with the number of GOPs per block.

Table 7 PJEM-GOP-RA: Parallel efficiency. Four People video sequence.

Number of processes	Number of GOPs per block 2	3	4
6	81.7%	88.6%	86.9%
8	79.1%	81.9%	86.7%
10	77.6%	79.1%	N/A

Table 8 shows the efficiency obtained by the PJEM-GOP-AI parallel algorithm using between 10 to 30 processes. As the number of processes exceeded the number of computing nodes in the test parallel platform used, several processes were mapped in the same computing node. Good efficiencies were obtained in all cases, and in several of them they were almost ideal. When the JEM reference software was used to encode using the AI mode, the computational cost was too high and, therefore, the computational cost of the manager process did not affect the global efficiency. With the efficiencies shown in Table 8, in most cases, we obtained very high, almost ideal speed-ups so the

coding times were reduced as many times as the number of coding processes, as long as the number of Blocks was sufficient to be distributed among all the processes. Furthermore, the manager process was able to balance the computing work because a work block assigned to each process was composed of a single frame. Similar conclusions can be drawn for Algorithm PJEM-GOP-RA, when the number of work blocks (GOP block) is high enough.

Table 8 PJEM-GOP-AI: Parallel efficiency.

Video sequence	QP	Number of MPI processes 10	16	20	24	30
PEOPON	37	99%	94%	94%	89%	92%
	32	99%	94%	94%	89%	92%
	27	99%	95%	94%	90%	92%
	22	98%	94%	94%	89%	92%
PKSCNE	37	98%	98%	99%	95%	90%
	32	97%	97%	94%	93%	91%
	27	98%	98%	98%	94%	91%
	22	98%	97%	96%	94%	89%
FOURPE	37	99%	98%	98%	96%	91%
	32	99%	98%	98%	98%	92%
	27	99%	98%	99%	96%	92%
	22	98%	98%	97%	95%	91%
PARTYS	37	99%	98%	99%	96%	91%
	32	99%	93%	99%	95%	93%
	27	99%	94%	98%	95%	93%
	22	98%	98%	99%	95%	92%

6 Conclusions

In this paper, we described a new video coding framework called JEM. Due to the computational needs of this new encoder, we presented a JEM model parallel encoder specifically designed for distributed memory platforms for both All Intra (AI) and Random Access (RA) coding modes. Two pairs of algorithms for the coordinator and encoding processes were described for both coding modes. The experiments we conducted showed there was a close relationship between the number of processes in the coding process, the number of GOPs in a sequence, and the size of the block to be distributed to each process. After comparing sequential and parallel versions, the parallel approach was found to obtain a good computational performance for AI and RA encoding modes, but at a slight R/D performance cost. We can conclude that generally speaking, the greater the number of GOPs per block, the greater the efficiency. Moreover, the efficiency of parallel versions showed better scalability for the highest resolution video sequences, and we could use more parallel encoding processes in order to reduce the overall encoding time.

References

1. GCC, the gnu compiler collection. Free Software Foundation, Inc. http://gcc.gnu.org (2009-2012)
2. Alshina, E., Alshin, A., Choi, K., Park, M.: Performance of JEM 1 tools analysis. Tech. rep., JVET-B0044 3rd 2nd JVET Meeting:San Diego, CA, USA (2016)
3. Bjontegaard, G.: Calculation of average PSNR differences between RD-curves. Tech. Rep. VCEG-M33, Video Coding Experts Group (VCEG), Austin (Texas) (2001)
4. Chen, J., Alshina, E., Sullivan, G.J., Ohm, J.R., Boyce, J.: Algorithm description of joint exploration test model 3. Technical Report JVET-C1001 (2016)
5. Chen, J., Alshina, E., Sullivan, G.J., Ohm, J.R., Boyce, J.: Algorithm description of joint exploration test model 7. Technical Report JVET-G1001-v1 (2017)
6. Cisco: Cisco visual networking index: forecast and methodology, 2016-2021. Tech. rep. (2017). https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html
7. García-Lucas, D., Cebrián-Márquez, G., Díaz-Honrubia, A.J., Cuenca, P.: Acceleration of the integer motion estimation in JEM through pre-analysis techniques. The Journal of Supercomputing, https://doi.org/10.1007/s11227-018-2352-3 pp. 1–12 (2018)
8. Grois, D., Nguyen, T., Marpe, D.: Performance comparison of AV1, JEM, VP9, and HEVC encoders. pp. 10396 – 10396 – 12 (2018). DOI 10.1117/12.2283428. URL https://doi.org/10.1117/12.2283428
9. HEVC Reference Software, https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/tags/HM-16.3/:
10. ITU-T, ISO/IEC JTC 1: Advanced video coding for generic audiovisual services. ITU-T Rec. H.264 and ISO/IEC 14496-10 (AVC) version 16, 2012 (2012)
11. (JVET), I.J.V.E.T.: Algorithm description of joint exploratory test model (JEM). Tech. rep., First JVET meeting, Geneva (2015)
12. Karczewicz, M., Alshina, E.: JVET AHG report: tool evaluation (AHG1). Tech. rep., Technical Report JVET-D0001 (2016)
13. MPI Forum: MPI: A Message-Passing Interface Standard. Version 2.2 (2009). Available at: http://www.mpi-forum.org (Dec. 2009)
14. Ohm, J., Sullivan, G., Schwarz, H., Tan, T.K., Wiegand, T.: Comparison of the coding efficiency of video coding standards - including high efficiency video coding (hevc). Circuits and Systems for Video Technology, IEEE Transactions on **22**(12), 1669–1684 (2012)
15. Schwarz, H., Rudat, C., Siekmann, M., Bross, B., Marpe, D., T.Wiegand: Coding efficiency complexity analysis of JEM 1.0 coding tools for the random access configuration. Tech. rep., JVET-B0044 3rd 2nd JVET Meeting:San Diego, CA, USA (2016)
16. Sjoberg, R., Chen, Y., Fujibayashi, A., Hannuksela, M.M., Samuelsson, J., Tan, T.K., Wang, Y.K., Wenger, S.: Overview of HEVC high-level syntax and reference picture management. IEEE Transactions on Circuits and Systems for Video Technology **22**(12), 1858–1870 (2012). DOI 10.1109/TCSVT.2012.2223052
17. Sullivan, G., Ohm, J., Han, W., Wiegand, T.: Overview of the high efficiency video coding (HEVC) standard. Circuits and systems for Video Technology, IEEE Transactions on **22**(12), 1648 –1667 (2012)
18. Sze, V., Budagavi, M., Sullivan, G.: High efficiency video coding (HEVC). Springer (2014)