IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2024.0429000

Performance, limitations, and design issues of the integration of a hardware-based IME module with HEVC video encoder software

OTONIEL LÓPEZ-GRANADO[1], HECTOR MIGALLÓN[1], ESTEFANÍA ALCOCER[1], ROBERTO GUTIERREZ[2], GLENN VAN WALLENDAEL[3], and MANUEL P. MALUMBRES[1] (Member, IEEE)

[1]Computer Engineering Department, Miguel Hernandez University of Elche, Elche 03202 Spain (e-mail: otoniel, hmigallon, ealcocer, mels@umh.es)
[2]Communication Engineering Department, Miguel Hernandez University of Elche, Elche 03202 Spain (e-mail: roberto.gutierrez@umh.es)
[3]Department of Electronics and Information Systems, 9052 Technologiepark-Zwijnaarde 19 Blegium, Ghent University (e-mail: glenn.vanwallendael@ugent.be)

Corresponding author: Otoniel López-Granado (e-mail: otoniel@umh.es).

This research was supported by Grant PID2021-123627OB-C55, funded by MCIN/ AEI/ 10.13039/ 501100011033 and by 'ERDF A way of making Europe'.

ABSTRACT High Efficiency Video Coding (HEVC) was designed to improve on its predecessor, the H264/AVC standard, by doubling its compression efficiency. As in previous standards, motion estimation is critical for encoders to achieve significant compression gains. However, the cost of accurately removing temporal redundancy in video is prohibitive, especially when encoding very high resolution video sequences. To reduce the overall video encoding time, we have proposed the implementation of an HEVC motion estimation block in hardware, which can achieve significant speed-ups. However, when the IP hardware is integrated into a software platform, there are several constraints and limitations that reduce its impact on the overall encoding time. In this paper, we analyse these issues in detail to identify the main bottlenecks of the overall software/hardware encoding system. From this analysis, we propose a final integration of the hardware motion estimation module with a hardware unit combined with the slice-based parallel version of the HEVC encoding software. The resulting integrated version is able to achieve the best performance in terms of global speed-up, up to 149.63x compared to the sequential version of the HEVC encoder using the full search motion estimation algorithm.

INDEX TERMS Video coding, HEVC, FPGA, Integer motion estimation, Inter prediction, SAD architecture, Asymmetric partitioning

I. INTRODUCTION

THE The Joint Collaborative Team on Video Coding (JCT-VC) launched the High Efficiency Video Coding (HEVC) standard [1] to replace the previous H.264/AVC [2] standard to cope with the demands of the audiovisual and entertainment industry by, for example, providing support for ultra-high definition (UHD) content with high dynamic range (HDR) and high frame rate (HFR) extensions, among others. The HEVC standard improves the coding efficiency with respect to its predecessor, H.264/AVC (high profile), delivering the same video quality with just half of the required bit rate [3]. Although HEVC is not the latest video standard to be released, it is the de facto industry standard. For example, Digital Terrestrial Television (DTT) broadcasts in both Europe and the US are based on HEVC, albeit with different standards. More recently, Twitch, a streaming platform primarily focused on video games, announced at TwitchCon in San Diego in September 2024 that it would implement HEVC for all of its users in an attempt to improve the quality of real-time video broadcasts, which is the aim of this work.

In terms of complexity, the HEVC encoder is much more complex than the H.264AVC encoder [4], and as in the previous standard, motion estimation (ME) is by far the most computationally intensive encoding tool, consuming about 90% of the total encoding time [5]. The huge increase in complexity compared to its predecessor is mainly due to an increasing number of Coding Tree Unit (CTU) partitioning modes. Thus, taking into account the increased number of potential reference frames and the computational complexity of the HEVC variable block size motion estimation (VBSME) module, performing motion estimation with the HEVC encoder is a computationally demanding task that needs to be offloaded in order to work in practical applications. To reduce the overall complexity, two approaches can be used

based on the design of dedicated (a) hardware or (b) software accelerators.

Most state-of-the-art hardware accelerator proposals focus on speeding up the ME process to reduce the overall complexity of the encoder by means of field-programmable gate array (FPGA) accelerators. In HEVC, the module responsible for the motion estimation process is called the integer motion estimation (IME) module. The core of IME is based on a motion search algorithm, the full search (FS) algorithm, which is the one that achieves the best performance using a greedy approach. This algorithm searches for the motion of a given block or prediction unit (PU) at all positions of a predefined search area in a given reference frame. Thus, the result of the FS algorithm will be the closest block to the original in the reference frame. The resulting candidate will minimise the residual error (minimum bit rate) and provide the corresponding motion vector (MV).

Although the FS algorithm is computationally expensive in software designs (there are faster proposals that obtain a similar performance), there are many proposals that use this algorithm in the design of hardware accelerators due to the repetitive patterns and well-identified processing tasks within the IME process. For example, the authors of [5]–[11] present IME hardware modules based on the FS algorithm. In [5], the authors propose a sum of absolute differences (SAD) unit in a FPGA device that is able to check all the partition modes of a CTU except the asymmetric ones. The processing capabilities of their design allow this approach to run as fast as 30 frames per second (fps) for 2K video formats when the search area is reduced below the size recommended by the standard. Another IME hardware accelerator is proposed in [7]. In this work, all partitioning modes are supported but with a search area size reduction to ± 23 pixels. This approach is able to run at 30 fps with HD video formats. In [8], the authors propose a hardware architecture that supports all partitioning blocks with search areas of up to 256x256 pixels and analyse the impact of the search area size on the IME module performance. They showed an implementation running at 57 fps with a 720p video resolution, using a 64x64 search area and only one reference frame. In [10], the authors present a hardware architecture for IME computation using 16x16 pixels CTUs and a search area of ± 16 pixels. The authors integrated their IME architecture into Kvazaar HEVC encoder software running in a standalone board supported package (BSP) methodology [12]. In [11] authors propose IME and ASIC architecture for a fast hybrid pattern search algorithm for 32x32 CTUs that uses fixed search patterns (Square, Hexagonal) and local refinement patterns (two, three, and four-pointed) with a limited and fixed number of search points, reducing hardware complexity. It starts from the center, expanding outward in grids, and applies early termination beyond distance four, eliminating raster scans.

In [13] and [14], the authors present hardware implementations of other suboptimal motion search strategies called fast ME algorithms, such as diamond search (DS) or TZSearch (TZS). In [13], two hardware DS algorithms (sequential and parallel) for a CTU size of 64x64 and a search area of 144x144 pixels are proposed. The parallel version is able to compute eight SADs corresponding to each block position in the DS at the same time. That architecture is able to compute a full-HD resolution video sequence at 30 fps running at 198.733 MHz. However, this implementation does not support asymmetric partitioning. In [14], the authors present a hardware implementation of the TZSearch algorithm. The main advantage of this architecture is the small area used in the FPGA, but at the cost of not supporting asymmetric partitioning. In [15], the authors proposed a memory-aware fractional ME (FME) architecture supporting only four PU sizes for the HEVC video coding standard. The architecture was described in VHDL and the synthesis results were obtained for 45nm Nangate standard ASIC cells, achieving 60 frames per second at 2160p video resolution.

In [16] and [17], the authors propose a new hardware architecture that implements the IME module of the HEVC encoder using the FS algorithm. Their design provides two innovative techniques: (a) a new SAD adder tree structure and (b) a special memory scan order. The proposed design is able to achieve encoding speeds of 116 fps and 30 fps with 2K and 4K video formats, respectively. In [18] author proposes a similar IME architecture for HEVC encoder achieving 10fps at 4K resolution. The proposed architecture organizes on-chip memory into Horizontal RAM, Vertical RAM, and a Row Buffer to enable efficient data reuse. This structure minimizes external memory access and supports continuous Full Search scanning with reduced latency and hardware cost. Recently, in [19], the authors present a ME hardware for the versatile video coding standard (VVC) using 64x64 systolic array and using the same snake scan order than the one proposed in [17] and an adder tree block. The proposed hardware is able to compute 30fps for 1080p resolution for a CTU size of 128x128 pixels, reusing Motion Vector (MV) from the 64x64 ones and a search area of 128x128 pixels. In [20], the authors proposed an efficient hardware architecture for integer motion estimation (IME) in the Versatile Video Coding (VVC) standard, targeting the high computational complexity associated with large search areas and variable block sizes. Its design introduces two key innovations: a data reuse strategy based on overlapping 8×8 sub-blocks across search points, and an early termination mechanism using thresholds derived from previously computed SAD values. The early termination strategy reduces the motion estimation a 70%, reducing the computational overhead. The architecture supports all PU sizes defined in VVC and can process 8K (7680×4320) resolution video sequences at 60 frames per second, operating at 182.99 MHz. In [21], the authors introduced an interpolation-free FME algorithm for VVC, designed to reduce hardware complexity while maintaining coding efficiency. By using a rate distortion-based error surface model and eliminating iterative interpolation, their hardware achieves 8K@60fps throughput. In [22], the authors developed a high-throughput hardware architecture for affine motion estimation in VVC based on an iterative refinement method with parallel pipelin-

ing. The design integrates motion vector adaptation and edge-based gradient analysis to improve prediction accuracy while maintaining low latency. It supports 4K@60fps real-time processing using a 64x64 search window size.

Many of these works that propose the computation of IME on an FPGA using a full search algorithm, especially those related to the VVC standard, introduce early termination techniques to reduce the computational cost, thus trying to make the FPGA implementation feasible. Moreover, none of these studies perform a practical feasibility analysis using OpenCL or similar tools. However, some contributions as in [23] and [24] have conducted comparative analyses of IME implementations developed as GPU kernels and those implemented as FPGA IPs, highlighting the big differences between both approaches in terms of energy efficiency.

Furthermore, in [25], the authors propose an approach that utilises both CPU and GPU resources via OpenCL. In [26], the authors investigate the feasibility of using OpenCL as an alternative to VHDL for FPGA development, evaluating an implementation of the Block Matching Motion Estimation module. They conclude that OpenCL results in greater resource and performance overheads than low-level HDL-based implementations.

Another way to reduce the overall complexity of the HEVC encoder is through software acceleration techniques (i.e. multithreading). There are many previous works concerning HEVC software accelerators [27]–[35] that range from the acceleration of a specific coding tool [28], [34], [35] to the acceleration of the overall HEVC encoding process [27], [29]–[31]. We will focus on the latter set of acceleration approaches.

In particular, we will exploit the intrinsic spatial parallelism of the HEVC video encoding process by dividing each video frame into slices. Each slice will consist of a set of consecutive CTUs in scan order. The idea is to encode each frame using one thread per slice. Thus, assuming that the encoding of one slice does not depend on the others, all slices of a frame would be computed at the same time. This approach is also known as slice-based parallel encoding. In a previous work, we proposed a slice-based parallel version of the HEVC encoder [30] which is able to work with different encoding modes to achieve speed-ups of up to $9.3x$ and $8.7x$ using a 12-core parallel platform for the all intra (AI) and random access (RA) coding modes, respectively.

After analysing the related works of the state-of-the-art, we noticed that none of them analyse the performance of their IME hardware proposals when integrated into the video encoder software (HEVC, VVC, etc.) showing the effects of integration as API and data transfer overheads. Indeed, most of the proposals only describe and evaluate the HW unit alone without taking into account their performance when integrated with a software video encoder. So, no details are given about memory transfers between HW device and CPU, the overhead introduced by APIs like OpenCL, synchronization issues, etc.

In this work, we propose an HEVC encoder that integrates both hardware (HW) and software (SW) acceleration techniques to analyse the behaviour of the overall video coding system, identifying the main performance limitations found in the integration process. First, we will deploy up to four HW units, which are spatially replicated on an FPGA board, to accelerate the HEVC IME coding tool. This allows all HW units to be used simultaneously (independent hardware resources). Second, we will use several threads to encode each frame using our slice-based, parallel version of the HEVC reference software. Once the hybrid HEVC accelerated encoder has been developed, we will first analyse the acceleration provided by HW and SW alone to expose the benefits and limitations found. Based on these results, we will then determine the optimal configuration for the final HW/SW integrated accelerator through performance testing. To the best of our knowledge, this is the first study to fully integrate the HEVC video encoder with both hardware (FPGA) and software (parallel slice-based) accelerators.

The main contributions of this work are the following. Firstly, we integrate and evaluate an FPGA-based integer motion estimation (IME) hardware module within a standard HEVC encoder. This addresses a gap in prior research, which has typically overlooked the impact of software-hardware interaction. Secondly, we identify and quantify key limitations that arise from real-world integration, such as memory transfer delays and OpenCL communication overheads. Third, we explore hardware parallelisation strategies using multiple IME units and analyse their effectiveness within the slice-based parallelism of the HEVC encoder. Our findings provide new insight into the performance and scalability challenges of deploying hardware acceleration in practical video coding environments.

The rest of the paper is organised as follows. In Section II, we define the proposed hybrid software/hardware video accelerator architecture, explaining how the integration of both HW and SW was done after evaluating their performance behaviour. Section III describes the experimental tests designed to evaluate the final, fully integrated hardware/software version of the proposed method. Finally, in Section V, some conclusions and future work are discussed.

II. PROPOSED SOFTWARE-HARDWARE VIDEO ACCELERATOR ARCHITECTURE

In this section, we will describe the proposed HEVC integrated accelerator architecture, explaining both the hardware and software accelerators and showing their potential benefits and limitations. Then, we will describe the integration framework driven by OpenCL [36], where we show how HEVC software is able to initialise the HW platform based on Xilinx FPGA boards and invoke the HW kernel process to perform IME operations. Finally, having analysed the behaviour of the hardware and software approaches, we will determine the optimal configuration for the HEVC integrated accelerator, which will be evaluated in the next section.

One of the main novelties included in HEVC is the quad-tree structure for picture partitioning, known as Coding Tree

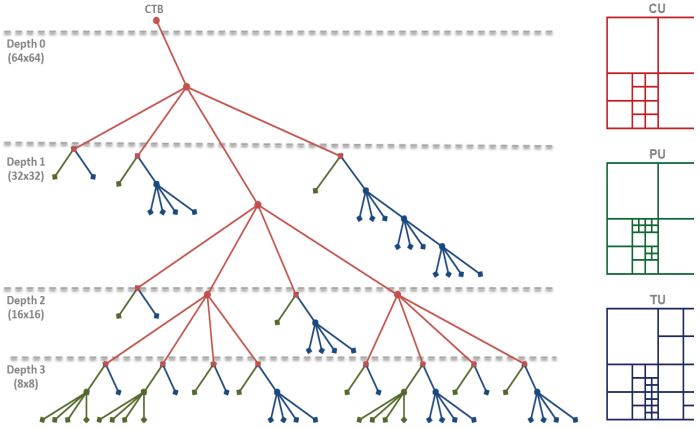


FIGURE 1. Relationship between CUs, PUs and TUs.

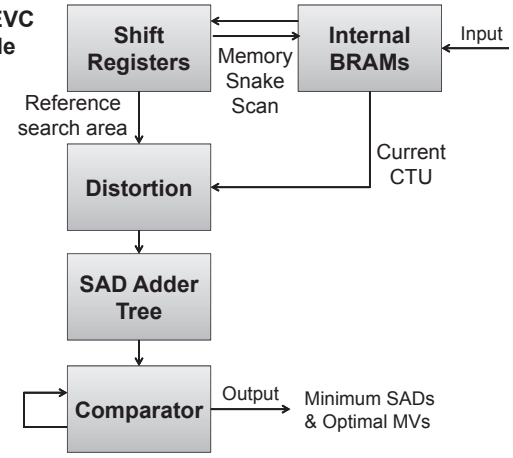


FIGURE 2. Hardware IME module.

Unit (CTU) [3]. This structure can be partitioned further into coding units (CUs), prediction units (PUs), and transform units (TUs) (see Fig. 1). PUs are the elements that store the prediction information, such as motion vectors (MVs). PU sizes can range from 64x64 to 8x8, either symmetrical or asymmetrical. HEVC defines eight possible partitions for each CU size, including square partitioning (2Nx2N and NxN), vertical and horizontal splitting (2NxN and Nx2N), and asymmetric splitting, where the CU is divided into two rectangular areas of sizes 1/4 and 3/4 in each of the four directions (2NxnU, 2NxnD, nLx2N, and nRx2N). A further description of HEVC CTU partitioning can be found in [37]. Each time a partition is tested for a particular candidate CU, a motion estimation process (IME operation) is performed to determine its rate distortion (R/D) cost. A 64x64 pixel CTU can be partitioned into a large number of possible partitions, all of which must be evaluated to determine the CTU partitioning scheme that provides the best R/D cost. Consequently, the overall complexity of the CTU motion estimation process is too high, which is the motivation for designing faster approaches for the partition process and low-complexity IME alternatives.

A. IME HARDWARE MODULE DESCRIPTION

Here, we provide an overview of our IME hardware design. It is based on the FS algorithm (the one offering optimal motion search performance), and it is able to evaluate all possible partitions of a CTU to provide the final partitioning set with the best R/D performance. The IME module consists of (a) two internal memory areas, one for storing the pixels of the current CTU and the other for storing the search area pixels from the selected reference frame, (b) a memory area for storing the difference between current and predicted blocks (residual error), (c) a SAD adder tree unit, and (d) a comparator unit that stores the minimum SAD and the corresponding MV of every single partition block, as shown in Fig. 2. Further details regarding the architecture and functionality of the base IME module can be found in [17].

In this work, the proposed IME hardware module has been configured to operate with CTUs of 64x64 pixels and a maximum search area size of 192x192 pixels which corresponds to a search range of 128x128 pixels. We have used a Xilinx Alveo U280 FPGA card from Xilinx, specially suited for datacenters, to accommodate up to four IME units that are able to work at the same time (by using different memory banks and direct memory access (DMA) channels). The Xilinx Alveo U280 FPGA accelerator card includes 8 GB HBM2 and 32 GB DDR4 external memory, a 16-lane PCI Express, and a custom-built UltraScale+ XCU280 FPGA device. The XCU280 FPGA uses AMD stacked silicon interconnect (SSI) technology to increase the density by combining multiple super logic regions (SLRs) into one device. An SLR is a physical section of the FPGA with a specific amount of resources and connections. Fig. 3 shows the SLRs and the connected external devices. The Ultrascale+ XCU280 FPGA comprises three SLRs, with the bottom SLR (SLR0) including a high bandwidth memory (HBM) controller to interface with the HBM2 subsystem through 32 pseudo-channels (PCs) each with direct access to 256 MB of storage (8 GB in total). Each 256-bit PC operates at 450 MHz, yielding a maximum bandwidth of 14.4 GB/s. The full system can thus achieve a theoretical bandwidth of 460.8 GB/s. The bottom SLR also connects to 16 lanes of the PCI Express (PCIe) that can operate at up to 16 GT/s (Gen4). Both SLR0 and SLR1 connect to a 64-bit, 2400 MT/s DIMM with 16 GB DDR4 and error correcting code (ECC), for a total of 32 GB of DDR4. Table 1 lists the allocation of memory resources and the available resources for each SLR.

In Fig. 4, a diagram of the memory banks assigned to each HW unit is shown. As can be seen, there are HBM banks that act as an interface with the PCIe-DMA host channels to accommodate data transfers between the host (CPU) and HW unit (FPGA). The HW unit will receive both CTU and SA pixels in the corresponding input HBM memory buffer (Rd ptr) to perform the motion estimation operation. Thus, when the read operation is signalled, the HW unit will start to

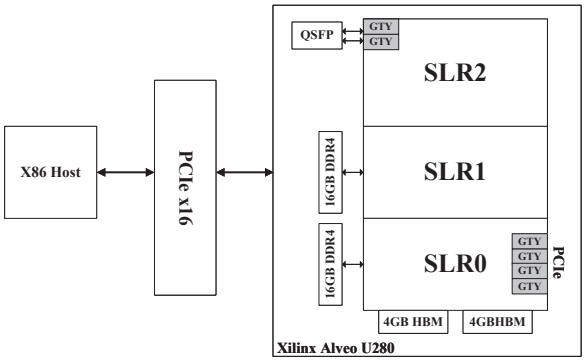


FIGURE 3. XCU280 floor-plan with super logic regions and external memory devices.

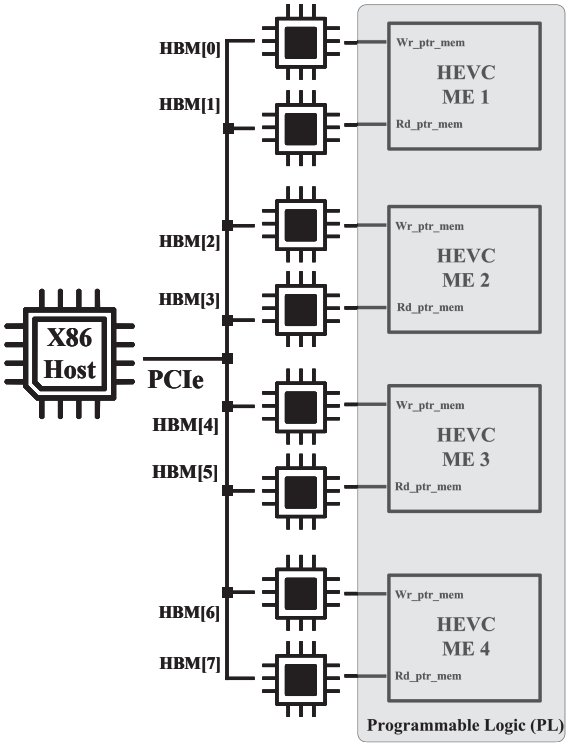


FIGURE 4. Memory distribution of hardware ME units.

copy the CU and SA pixels to its own Block-RAM36 memory areas. Then, full search motion estimation is carried out to compute the SAD and MV of each possible PU, and they are stored in the corresponding Block-RAM36 memory area. In order to complete the IME operation, the HW unit will copy the final result into the output HBM buffer (Wr ptr), signalling the write operation to start the corresponding DMA transfer to the host. In Table 2 we show the FPGA resources required by the implementation of four different Intellectual Property cores (IPs) that will map one, two, three, or four HW units. Although four HW units can fit in the U280 board (there is enough space), there is a limitation imposed by the FPGA

TABLE 1. Platform resource availability for each SLR.

Resources	SLR0	SLR1	SLR2
HBM2	32x256MB	-	-
DDR4	16GB	16GB	-
CLB LUT	386K	364K	381K
CLB register	773K	729K	763K
Block-RAM36	600	576	600
Ultra RAM	320	320	320
DSP-48	2664	2784	2856

board design that splits the overall implemented silicon area into two different SLRs. Thus, we have to map up to two HW units in SLR0, so the last two HW units should be placed in SLR1. This implies some overhead penalties for the HW units allocated in SLR1, since accesses to the HBM buffers need to cross SLR0 area.

TABLE 2. Resource utilisation of implemented hardware unit.

Resources	1 HW Unit	2 HW Units	3 HW Units	4 HW Units
CLB LUT	175K(15%)	350K (30%)	525K (46%)	701K (61%)
CLB register	174K	350K	524K	699K
Block-RAM36	48 (2.7%)	96 (5.4%)	144 (8.1%)	192 (10.8%)
Power (W)	8.5	10.5	11.9	12.9
Freq. (MHz)	249	247	192	176

B. SLICE-BASED PARALLEL HEVC ENCODER

The HEVC standard enables a video frame to be divided into a set of consecutive CTUs (slices). Each slice is configured independently (i.e. there is no data dependency between slices), enabling all the slices in a frame to be encoded simultaneously.

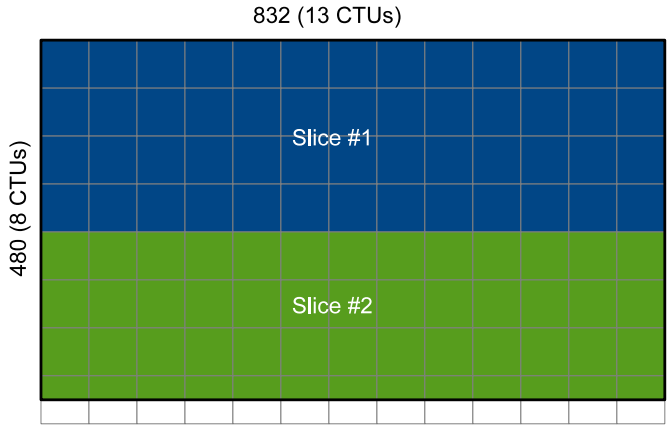
Each slice contains the same number of CTUs, except for the final slice, which may contain fewer CTUs if the total number of CTUs is not a multiple of the number of slices (see Fig. 5).

Each slice contains a data header with specific coding parameters about the slice (starting and ending CTU id, QP value, etc.). This extra information affects the compression performance since slice headers represent a bitstream overhead that reduces the overall compression ratio. In our proposal, the number of slices in the parallel algorithm is determined by the number of available encoding processes (threads). Each encoding process calculates the location and size of its corresponding slice (i.e. the start and end CTUs in the frame) following a static allocation scheme. The final slice will either equal or be smaller than the others.

This slice partitioning aims to achieve a balanced computational load, assigning to each process the same (or a similar) amount of data. Depending on the video sequence resolution to be encoded, there may be CTUs at the right-hand or bottom edges of a frame with fewer than 4096 (64×64) pixels. Figs. 5a and 5b show two different partition schemes for encoding an 832×480 pixel video sequence, where the total number of CTUs is 104 (13×8). Fig. 5a shows partitioning into two slices of 52 CTUs each, while Fig. 5b shows partitioning into six slices, where the first five slices

contain 18 CTUs each and the last slice contains 14 CTUs. In the last slice, only the first CTU has 4096 (64×64) pixels, and the remaining 13 CTUs have only 2048 (64×32) pixels.

The encoding process for each video frame is described below: (a) The current frame is loaded into shared memory; (b) each process immediately starts encoding its assigned slice once the frame has loaded; (c) once one process has finished encoding its slice, it waits for the others to finish; (d) once all processes have finished encoding their slices, the output bitstream is written in order. Further details regarding the architecture and functionality of the proposed slice-based algorithm can be found in [38].



(a) Partitioning into two slices

(b) Partitioning into six slices

FIGURE 5. **Slice partitioning of an 832 × 480 frame.**

C. OPENCL INTEGRATION

In this subsection, we describe the integration of both accelerators: the IME hardware and slice-based parallel coding accelerators. First, we will determine the communication framework between the host device (CPU) and the hardware device (FPGA). We will use the OpenCL [36] framework to allow the HEVC encoder to use our IME hardware module implemented on the FPGA device as a generic co-processor. OpenCL defines an API that allows programs running on the host to launch kernels on the computing devices and manage their memory, which is (at least conceptually) separate from the host memory.

Thus, on the one side, we have the slice-based HEVC parallel encoder version, defined in Section II-B, where each process (thread) will compute a single slice of one frame. One slice is composed of a predefined number of CTUs that will be encoded in raster order. When a single CTU is encoded, just before starting the encoding process, the HEVC encoder will send to the IME Hardware module the CTU and SA pixels of the selected reference frame to perform the FS motion estimation. As a result, the IME hardware module will provide the SADs and MVs of all possible partition blocks to the HEVC encoder, and they will be saved in a lookup table for later use. At this point the HEVC encoder starts the CTU encoding process with a recursive approach to exhaustively test the different partitions in order to find the partition set that provides the best R/D performance. During this high-resource-consumption process, every time that an IME operation is required for a particular partition block, we will obtain its corresponding MV and SAD from the lookup table where IME results were previously stored. As a consequence, we save a huge amount of computation time by avoiding calling the software IME process during the recursive CTU encoding process.

Algorithm 1 Invoking IME operation from HEVC encoder

```
1: procedure ComputeSad (CTU, SA)
2: n = GetHwUnit ( ) // Find one free HW unit
3: inputOCLBuffer[n] = CTU + SA // Copy CTU and SA
   pixels to the OpenCL input buffer of HW unit 'n'
4: EnqueueMem (inputOCLBuffer[n]) // Map input buffer
   into the FPGA device buffer (HBM) of HW unit 'n'
5: EnqueueTask (n) // Execution of IME in HW unit 'n'
6: EnqueueMem (outputOCLBuffer[n]) // Map output
   buffer into the FPGA device buffer (HBM) of HW unit
   'n'
7: WaitForEvents ( ) // Wait until all enqueued operations
   are done
```

On the other hand, our IME hardware module, introduced in Section II-A, may implement up to four independent IME hardware units that are able to work in parallel. Each hardware unit is composed of internal input buffers for storing the CTU and SA pixels, and an output buffer for storing the SADs and MVs of all possible partitions. The communication between the software (CPU) and the IME hardware module (FPGA) is driven by the Xilinx DMA subsystem (XDMA) of the FPGA device card. XDMA is used in conjunction with the PCIe IP block to provide high-performance data transfer between the host memory and the card's DMA subsystem.

In order to describe the SW/HW integration with OpenCL, we have modified the HEVC encoder to include a new software module that performs all the required functionalities related to the OpenCL framework, including initialisation, data transfer, and IME HW execution. Furthermore, some additional buffers are required to properly accommodate the

input (CU and SA pixels) and output data (SAD and MV lookup table) used in every CTU operation. In Algorithm 1, we show the pseudo-code associated with a single IME request at the beginning of CTU processing, as described before. As can be seen, each HW unit of the IME IP module will have its own input/output buffers on both the host (CPU) and device (FPGA) sides, previously allocated in the OpenCL initialisation procedure. As HW units have independent buffers, the software may launch, at the same time, as many IME operations as there are HW units available. Thus, if the IME module only has one HW unit and the slice-based encoder uses four slices/threads that compete for the HW unit, the result will be serialised by the OpenCL framework. However, if we have an IME IP module with four HW units, the threads do not have to wait for their IME requests. Thus, we achieve an additional hardware parallelisation when more than one HW unit is available in the IME IP module.

Notice that our slice-based HEVC parallel encoder uses as many threads as there are slices to encode a frame and our IME IP module may execute as many IME operations as there are HW units at the same time. It would be interesting to evaluate the behaviour of both the SW and the HW accelerators working alone to determine their limitations and define the best configuration for the integrated version. In the next section, we present the performance of the SW, HW and integrated versions.

III. NUMERICAL EXPERIMENTS

In this section, we show the experimental tests that we have designed to properly evaluate the HW/SW integration. We have considered the HEVC Random Access (RA) coding mode, where the first frame is encoded as an intra (I) picture frame and the rest of the frames of every GOP (Group of Pictures) are encoded as Inter picture frames (B) using reference frames from the past and future with respect to the coding order. Additionally, we have fixed the search area range (SA) size to the one established by the HEVC standard ($128x128$ pixels) and tested our proposal over all the video sequences suggested by the HEVC common conditions video set. All the tests were carried out under HEVC HM 16.3 software reference model [39], which was running on a shared memory parallel system equipped with two Intel(R) Xeon(R) Gold 6230 CPU @ 2.10GHz processor with 20 cores each and 256GB RAM installed. The operating system was CentOS Stream 8 with kernel Linux 4.18.0-305.25.1.el8_4.x86_64. The 8.5.0 gcc compiler version, the 4.5 OpenMP version and the Xilinx Runtime 2.15.225 version were used. As mentioned above, the system's FPGA is the Xilinx A-U280-A32G-DEV-G.

The experimental tests will be organised as follows.

- First, we will evaluate a single IME HW unit by (i) measuring the time required for both the input/output data transfers and the CTU computation operations, (ii) estimating the temporal cost of each pipeline stage of our IME HW proposal, and (iii) estimating/measuring the software overhead introduced by the OpenCL API.
- After determining the performance behaviour of an IME HW unit alone, we will study the integrated SW/HW version by (i) evaluating the slice-based parallel version with only one IME HW unit, analysing the impact on performance as the number of threads increases, and (ii) evaluating the behaviour of multiple IME HW units (two, three, and four) in order to find potential bottlenecks, overheads or limitations that may suggest the most appropriate number of HW units.
- Additionally, we will perform a comparative study of our IME HW unit and other SW motion estimation algorithms in terms of the computational cost. We will obtain the speed-ups provided by our HW approach and the differences in terms of the coding performance.
- Finally, taking into account the previous evaluation results, we will evaluate the proposed SW/HW integrated version of the HEVC encoder using an IME HW with one HW unit to encode all the video sequences from the HEVC Common test conditions (CTC) set.

A. PROFILING OF AN IME HARDWARE MODULE

First, we performed a time profile of the proposed IME HW module so as to measure the input/output data transfer operations, the IME HW computing times and the overhead introduced by the use of the OpenCL API. In Fig. 6, we can see the time profile obtained from the Vitis Analyzer tool during the encoding of one CTU of the Cactus video sequence. In the left panel of Fig. 6, we can see the different chronogram lines related to the execution of OpenCL API calls (General and queue operations), Data Transfer (read and write operations) and Kernel Enqueues (kernel execution operations). As shown in algorithm 1, when a HW IME operation is requested, three enqueue operations are launched in the following order (a) enqueue the transfer from host input buffer to FPGA (*clEnqueueMigrateMemObjects*), (b) enqueue kernel execution order (*clEnqueueTask*), and (c) enqueue the transfer from FPGA output buffer to the host (*clEnqueueMigrateMemObjects*). After enqueueing these operations, we have to wait until their completion by means of the *clWaitForEvents* function. Meanwhile HEVC encoder waits for completion, the operations are properly executed by the OpenCL scheduler, as shown in Fig. 6, beginning with the CTU and search area data transfer to the FPGA, continuing with the kernel execution and finishing with the transfer of the SADs and MVs from FPGA to the host memory buffer. For each CTU in a frame, this cycle of operations is repeated as many times as the number of available reference frames (different search areas).

In Fig. 7, the pipeline processing scheme of the IME hardware module is shown. At first place, before starting the computation process, we have to copy both CTU and SA pixels from HBM to the FPGA memory. This operation requires one clock cycle per pixel (40577 clocks). Then, the internal processing to compute both SAD and MVs requires 16462 clocks. This value breaks down as follows: (a) 64 clock cycles to load the shift registers with the CTU pixels, (b) 14 clock

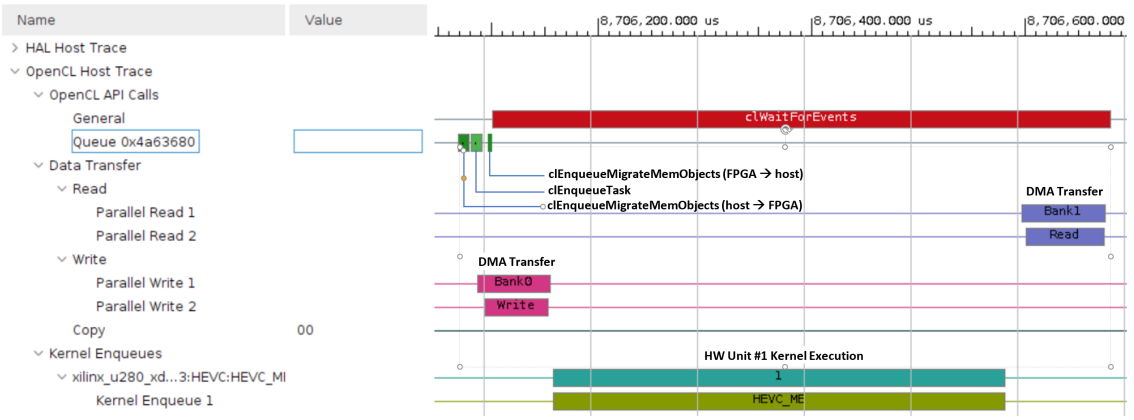


FIGURE 6. **Vitis Analyzer profiling for Cactus video sequence using one IME HW unit (coding of one CTU).**

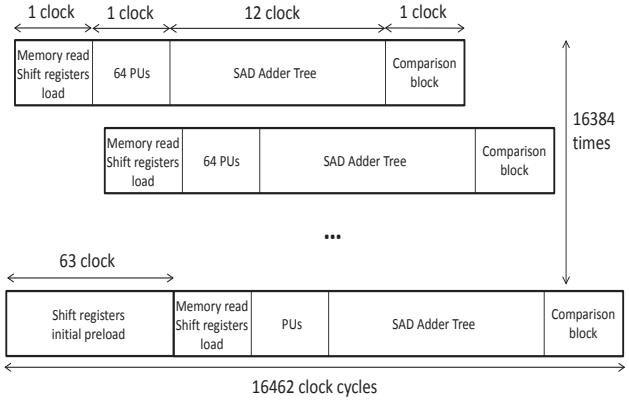


FIGURE 7. **Pipeline process of the IME hardware module architecture.**

cycles to compute: the SAD between CTU and the SA block at the first SA location (1 clock cycle), the execution of the adder tree (12 clocks cycles), the comparator stage to store the best SADs with their motion vectors (1 clock cycle), and, after the pipeline is full, (c) every clock cycle will test a new position of the search area, so we will need 16,383 additional clock cycles to complete the IME operation. Finally, the results (SADs and MVs) stored in the FPGA memory should be sent back to the host, requiring one clock cycle to transfer the best SAD and MV of each potential partition block (593 clocks). Therefore, the total number of clock cycles to perform one IME operation will be: $40577 + 16462 + 593 = 57632$ clocks, which at a frequency of 249 MHz, the overall HW processing time of one CTU makes 231 μs. Using the Vitis Analyzer tool, we have confirmed that the total time required by the IME HW to perform the motion/estimation process of a single CTU is 231 μs on average.

B. ANALYSING THE OPENCL OVERHEAD

In order to measure the software overhead introduced by the OpenCL API, we will also use the Vitis Analyzer tool since it is able to register all the OpenCL activity during the HEVC encoding process. In particular, in Table 3, we summarise the computational time required by the OpenCL primitives involved in the SAD computation of every single CTU during the HEVC encoding of the first 200 frames of Cactus video sequence.

For each OpenCL primitive we show the number of calls (*#CTUs*), the total aggregated time (*Time*), and the average time per call (*OneCTU*). As shown in Algorithm 1, to perform a single *CTU_SAD* operation we need to (a) copy the CTU and SA pixels to the input buffer, (b) enqueue the input buffer (order DMA transfer to FPGA), (c) enqueue the Kernel execution order, (d) enqueue the output buffer (order the DMA transfer to host), (d) wait for the completion of all enqueued operations, and (e) release queueing resources and store the SAD and MV results from output buffer to the application buffer. So, taking into account the average time of all the OpenCL functions and the additional house keeping tasks (releasing resources and memory copy operations) required to compute the SAD of a single CTU, the application (HEVC encoder) will spend an average of 344 μs. So, if the IME HW model only requires 231 μs, the overhead introduced by the use of the OpenCL API will represent an additional 49%.

Finally, to get the whole picture about the OpenCL overhead, the time required to perform the DMA transfer operations between host and FPGA devices should be determined. Again, the Vitis Analyzer tool is able to provide detailed measures of the DMA memory transfers. The obtained results show that the DMA transfer from the host to HBM FPGA memory of the CTU and SA pixels requires 37 μs (40577 bytes), whereas the time required to get the SAD and MVs (DMA transference from HBM FPGA memory to the host) is 31 μs (4744 bytes). It should be note that as the size of the DMA transfer data increases, so does the DMA throughput, reducing the overhead of the DMA transfer protocol (as expected). As it can be seen in Fig. 6, the first DMA transfer (from host to FPGA) starts after enqueueing the input buffer, and the second DMA transfer starts just after finishing the kernel execution operation.

TABLE 3. OpenCL operation tasks.

API Name	#CTUs	Time (ms)	OneCTU (μs)
clEnqueueMigrateMemObjects (host-to-FPGA)	386,580	1,271.46	3.29
clEnqueueTask	386,580	1,325.20	3.43
clEnqueueMigrateMemObjects (FPGA-to-host)	386,580	283.75	0.73
clWaitForEvents	386,580	126,026.24	326.00

FIGURE 8. Vitis Analyzer profile of the first 16 frames of Cactus video sequence using an IME module with four HW units.

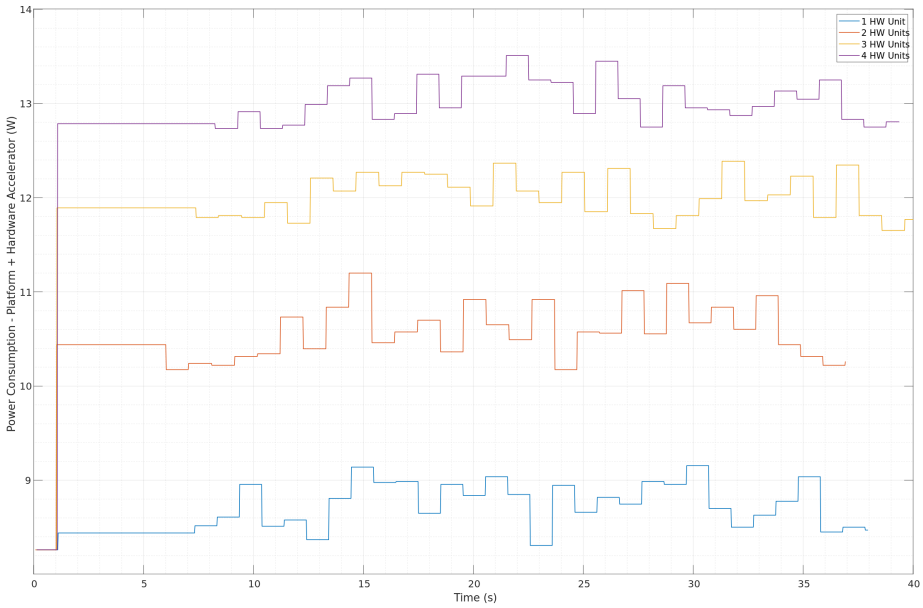


FIGURE 9. Power consumption profile when coding Cactus Video sequence using one, two, three and four IME HW units.

C. INTEGRATED VERSION EVALUATION

In order to determine the impact on application performance of our integrated HW/SW accelerator we will analyse the behaviour of using multiple hardware units in combination with one or more software threads. We will use our integrated version to encode the first 200 frames of Cactus video se-

TABLE 4. Time profile using different Threads and HW Units (Cactus video sequence).

HW Unit	Threads	Time(s)	CTU_SAD(μs)	IME (μs)	SW_SpUp	HW_SpUp	ΔCTU_SAD
HW1 (249 MHz)	1	3,379.5	344,035	0,108	1.00x	-	-
	2	1,773.0	359,766	0,136	1.91x	-	-
	4	962.8	388,388	0,185	3.51x	-	-
	8	547.8	443,075	0,239	6.17x	-	-
	12	413.6	530,569	0,286	8.17x	-	-
	16	337.2	654,109	0,343	10.02x	-	-
	20	301.1	788,649	0,403	11.22x	-	-
HW2 (247 MHz)	1	3,373.7	343,617	0,106	1.00x	-0.17%	-0.12%
	2	1,763.5	360,313	0,144	1.91x	-0.53%	0.15%
	4	961.3	384,845	0,187	3.51x	-0.15%	-0.91%
	8	548.9	442,305	0,236	6.15x	0.20%	-0.17%
	12	414.1	532,898	0,282	8.15x	0.12%	0.44%
	16	338.6	651,251	0,340	9.96x	0.42%	-0.44%
	20	299.0	779,760	0,395	11.28x	-0.71%	-1.13%
HW3 (192 MHz)	1	3,411.0	412,166	0,109	1.00x	0.93%	19.80%
	2	1,778.2	437,193	0,148	1.92x	0.29%	21.52%
	4	972.7	471,405	0,183	3.51x	1.03%	21.37%
	8	554.9	560,534	0,226	6.15x	1.29%	26.51%
	12	416.6	680,632	0,285	8.19x	0.73%	28.28%
	16	342.6	855,136	0,335	9.95x	1.62%	30.73%
	20	303.1	1051,822	0,408	11.25x	0.67%	33.37%
HW4 (176 MHz)	1	3,415.7	437,171	0,108	1.00x	1.07%	27.07%
	2	1,782.3	464,603	0,136	1.92x	0.53%	29.14%
	4	973.8	503,222	0,182	3.51x	1.15%	29.57%
	8	554.8	604,964	0,232	6.16x	1.27%	36.54%
	12	419.7	741,479	0,277	8.14x	1.48%	39.75%
	16	346.0	937,22	0,348	9.87x	2.61%	43.28%
	20	305.5	1166,093	0,396	11.18x	1.48%	47.86%

quence using the RA coding mode, a QP value of 32, and a combination of HW units (1 to 4) and software threads/slices (1 to 20). In Table 4, we show (a) the number of hardware units in the IME HW module and its operating clock frequency (b) the overall HEVC encoding time (*Time*), (b) the average time to perform a single CTU_SAD computation, (c) the average time to perform an IME operation over a single location of the SA, (d) the acceleration factor with respect to the HEVC encoding time with only one thread (*SW_SpUp*), (e) the acceleration factor with respect to the HEVC encoding time with only one HW unit (*HW_SpUp*), and (f) the acceleration factor with respect to the CTU_SAD time with only one HW unit (ΔCTU_SAD).

As can be seen, all integrated versions that use one, two, three, or four hardware units obtain good SW speed-ups (column *SW_SpUp*), which means that our slice-based accelerator behaves in the same way, independently of the available number of hardware units. However, when comparing the behaviour of using different numbers of hardware units with the same number of threads (column *HW_SpUp*) we find that using more than one hardware unit does not reduce the overall encoding time, what is an unexpected result. After performing a detailed analysis of these results, we believe there are at least two explanations: (a) all hardware units are used at the same time in very few cases (only when working with the first CTU of every frame of the video sequence), so hardware parallelism is not properly exploited most of the time, and (b) due to the implementation of the IP module in the selected FPGA board, since the first two HW units fit in the SLR0 area, meanwhile the third and fourth HW units are allocated in the SLR1 area. As shown in Fig. 3, only the SLR0 area has direct access to the HBM banks, so the hardware units allocated in other areas need extra cycles to access HBM memory.

The first reason is the most relevant one here, since it clearly justifies the low utilisation of hardware units. In Fig. 8, we show the behaviour of the HW4 IME module (4 IME hardware units) when the first 16 frames (just two GOPs) of Cactus video sequence are encoded. Looking at 'Kernels Enqueues' section (at the bottom of the figure), we can see that only for the first CTU of every frame all four IME units work in parallel. Remember that before start the encoding of a new frame all coding threads perform a synchronisation barrier to apply a filtering process just after decoding the slices of actual frame (since it will be used as a reference frame in the following). Additionally, we can also see that the use of two or three HW units in parallel occurs just very few times. This means that threads are quickly desynchronised as they code the CTUs of their slices, and therefore, most of the time only one hardware IME module will be used at the same time (ie. there is no hardware parallelism). At the other hand, in Table 4, we have included the average *CTU_SAD* to determine the average time required by the HEVC encoder to perform SAD computation of a single CTU when using different versions of the IME HW module. The ΔCTU_SAD column shows the relative increase in *CTU_SAD* with respect to that observed with a single HW Unit. Here we can conclude that these results are very similar when using 2 HW units, but increase significantly for three and four HW units. This effect is mainly due to the clock frequency drop for the design of 3 and 4 HW units as their design require the use of the SLR1 area, as mentioned above.

In Fig. 9, we show a comparison of the energy consumption of different IME designs with one, two, three and four HW units. Power consumption measurements (in watts) were

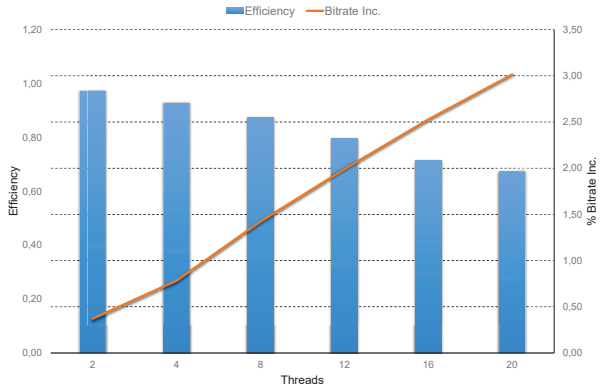


FIGURE 10. Parallel Coding efficiency and bitrate overhead for NebutaFestival video sequence.

taken using the Vitis analyzer tool while encoding Cactus video sequence using the HEVC integrated version. Note that the obtained measures correspond to the total power consumption of the U280 platform and the proposed IME HW designs. As expected, the IME design with only one HW Unit requires the least energy, being on average 8.5 W (see Table 2).

Thus, considering the observed software behaviour and the hardware implementation issues found, we recommend using the HEVC slice-based parallel version combined with only one hardware IME module (HW1). This setup is the one that provides the best trade-off among the speed-up, coding performance, power consumption and FPGA area usage.

Following the above analysis on the use of several hardware units, we proceed to perform an evaluation of the parallel slice-based version of the HEVC encoder with a single IME hardware module in terms of the speed-up, PSNR, and bitrate for all video sequences of the HEVC common test conditions (classes A+ to C). In Tables 6 and 7 we show the encoding time (*Time*), video quality (*PSNR*), and bitrate for all video sequences encoded with RA coding mode and a QP value of 32. As can be seen, the encoding time is drastically reduced as the number of threads increases. The maximum speed-up obtained was 13.5x, and it was obtained for the NebutaFestival video sequence using 20 threads. Regarding PSNR, there is a negligible penalty as the number of threads increases. For example in the BasketBallDrill test sequence, the PSNR is reduced by 0.2% in the worst case when 20 threads are used. A similar behaviour is obtained for the rest of the video sequences, where the maximum PSNR loss of 0.43% was obtained for the Kimono video sequence when 20 threads were used. However, the bitrate increment suffers a significant penalty as the number of threads increases, being the maximum bitrate increment of 44.35%, and it was obtained for the Johnny video sequence using 20 threads. As stated in [30], that increment is mainly due to a) the extra headers introduced on each slice, and b) the encoding performance losses caused by disabling the motion prediction across slice boundaries. Furthermore, CABAC (the HEVC entropy encoder) context models are not updated after the computation of each slice but after the frame computation. Thus, there is a trade-off between the acceleration and bitrate increment that will be determined depending on the final application requirements. We recommend using no more than eight threads, with four threads representing the most balanced proposal in terms of the acceleration (3.33x on average) and bitrate increment (4.82% on average). It is also worth noting that for high resolution video sequences (UHD and beyond), the penalty in the bitrate is highly reduced, as in the NebutaFestival sequence (see Fig. 10) where the maximum bitrate increment is 3.0%, whereas the acceleration efficiency is the best one, as it is possible to increase the number of threads for this kind of sequences.

D. COMPARISON AGAINST OTHER ME ALGORITHMS

We have also compared our IME HW ME proposal ($FS-HW1$) with other ME algorithms available in the HEVC reference software as (a) Full Search (*FS*), and (b) three fast ME algorithms based on the *TZSearch* strategy with different search patterns: diamond, selective, and enhanced diamond. In order to evaluate their ME performance, we have encoded 16 frames of the Cactus video sequence using the RA coding mode with a QP value of 32. The Cactus sequence with a resolution of 1920x1080 has 510 CTUs per frame and approximately 1479 PUs are evaluated per CTU, on average. In Table 8 we show the results of the SW Slice-based approach working with several ME algorithms in order to make a fair comparison between them. In particular, for each ME algorithm we show (a) the overall encoding time (*Time*), (b) the average ME time of one CTU (ME/CTU) [1], and (c) the speed-up of the *CTU_SAD* operation using as reference the FS algorithm.

It is important to take into account that when using our HW IME module, (a) the IME/PU operation is very fast since it consists on one memory access and very few computations, and (b) the ME/CTU should also include the time required to complete the CTU_SAD operation performed at the begging of each CTU encoding (see Table 4) for each reference frame.

As can be seen, the proposed HW IME proposal is the fastest one, speeding up the ME process by up to 679x when compared with the SW FS version. However, the other SW fast ME approaches are also competitive at the cost of reducing the ME accuracy and increasing the bitrate obtained by the Full Search versions.

Finally, we have performed a complete evaluation of the IME SW/HW integrated version by comparing it to the FS algorithm of the HEVC reference software with all video sequences of the HEVC common test conditions. In Table 9 we show the computing time, PSNR and bitrate of both the HEVC using FS ME and our integrated SW/HW parallel version using four threads/slices per frame and one HW

[1]Notice that the ME computing of one CTU may require up to four reference frames, so the ME process is repeated so many times as the number of reference frames needed. In general, most of the frames in the encoding process use 4 reference frames per CTU.

TABLE 5. Comparison of the proposed architecture with state-of-the-art works

Design	[5]	Proposed	[8]	[7]	[18]	[19]	[20]
Codec	HEVC	HEVC	HEVC	HEVC	HEVC	VVC	VVC
Technology	Virtex-7	Virtex-7	Virtex-5	Virtex-6	Virtex-7	Virtex-7	Virtex-7
CTU size	64x64	64x64	64x64	32x32	64x64	128x128	128x128
Search area	104x104	64x64	64x64	48x48	128x128	128x128	64x64
Clock (MHz)	458.7	247	150	110	323	253	182.9
AMP	No	Yes	No	Yes	yes	yes	yes
Throughput	2K@30fps	4K@30fps	720p@57fps	1080p@30fps	4k@10fps	4k@30fps	8k60fps
Flip-Flops	39901	144302	199682	19744	107862	182327	30600
LUTs	24957	188664	210158	55346	120141	145606	85070
Memory (kB)	44	36	1229	148	36	468	80.03
Power(W)	-	3.16	13.60	-	8.16	-	-

IME module. As can be seen, speed-ups of up to 149.63x are obtained with a negligible PSNR loss and an average bitrate increment of 6.1%. Additionally, if we focus on high resolution video sequences like NebutaFestival, Traffic, or PeopleOnStreet, the average bitrate increment is just 1.91%, with speed-ups of up to 136.45x. As previously stated, the configuration that uses one HW unit and four threads per frame shows the best trade-off between the encoding acceleration and R/D performance penalty.

E. COMPARISON AGAINST OTHER STATE-OF-THE-ART PROPOSALS

As previously mentioned, most state-of-the-art IME hardware proposals are evaluated in isolation without considering their integration into the reference encoder software. This means that practical issues such as API interactions and data transfer overheads are overlooked. The main results they provide are about the performance of the HW design alone. So, we may proceed to compare our HW IME proposal with the ones in the literature trying to use the evaluation setups as close as possible. For that purpose, we have emulated our IME HW module over a Xilinx Virtex 7 FPGA, because the majority of literature review use that technology. To accommodate our IME HW setup to the ones found in the literature, we have reduced the search area (SA) to 64x64 pixels. Also, we have taken into account only the time required to compute one CTU in the HW module (no data transfer overheads considered).

In Table 5, we compare our proposal with other ones from the literature by means of several features that determine the IME HW module setup (i.e. CTU and SA sizes, Technology, etc.) and its performance (ie, throughput, FPGA resources). The most complete FS HW proposals are the ones that include the computation of the asymmetric partitions (AMP) like our proposal and the ones in [7], [18], [19] and [20]. As can be seen, our proposal is highly competitive as it is able to achieve an encoding speed of 30fps for 4k videos using a search area of 64x64 pixels. However it requires a significant amount of FPGA resources in a similar way than the one proposed in [19]. Although the proposal presented in [20] obtains better results in terms of throughput, this is mainly due to the early termination algorithm of the motion estimation. However, none of the above state-of-the-art proposals integrates their IME into the encoder software.

To the best of our knowledge, there are hardly any FPGA-based IME integration in HEVC software using OpenCL. Specifically, the only work we have found that uses OpenCL to evaluate an integration of the motion estimation kernel is the one proposed by Castro et al in [26]. As reported in [26], an algorithm for block matching motion estimation based on the H.264 video coding standard that uses only fixed 16x16 blocks is presented and implemented on an Intel Stratix 10 FPGA using OpenCL. The performance analysis focuses solely on the kernel itself, disregarding its full integration within the encoder and the cost of transferring the 16x16 blocks to the FPGA. However, our proposal is based on the HEVC standard. It uses variable block sizes ranging from 8x8 to 64x64 and incorporates asymmetric partitioning across a 128x128 search area, as well as a significant number of partition units per CTU. In terms of performance, both approaches produce similar results, but a fair comparison cannot be made given the significant differences between them, as previously mentioned.

IV. DISCUSSION

After the evaluation and the analysis of the results obtained in previous sections, we summarise the main hardware and software limitations found.

- First, as described in Section III-A, one of the main bottlenecks found in the IME HW module design is the time required to perform memory transfer operations to copy both the CTU and the SA pixels from FPGA HBM memory to the internal FPGA memory banks (40577 clock cycles) and send the ME results (SADs&MVs) back to the HBM memory (593 clock cycles). As the time required to compute ME and obtain the SAD and MVs of all partitions of a given CTU requires only 16462 clock cycles, the memory transfer operations represent the 71% of the overall time that IME HW module requires to compute process one CTU. So, assuming an operating frequency of IME HW module of 249 MHz, from the 231 μs to perform the whole ME process of a CTU (including memory transfers), only 66 μs corre-

spond with the ME computation.

- Besides, the HEVC encoder uses the OpenCL interface to communicate with the IME HW module at the FPGA device. To compute the SADs&MVs for a single CTU with one reference frame, the HEVC encoder spends 344 μs, on average, as described in Section III-B. So, if the time required by the IME HW module is about 231 μs, the overhead introduced by OpenCL is 113 μs, what represents an extra overhead of 49%.

  The delay chain of computing operations is represented in Table 10, based on the analysis of the IME HW module and the OpenCL interface. As can be seen, of the total 344 μs only 66 μs (19%) are related to ME computation; 233 μs (68%) are related to DMA and FPGA memory transfers; and 45 μs (13%) are related to other operations, such as OpenCL scheduling and resource management (OpenCL events, HEVC input and output buffers, etc.).
- When comparing the performance of the different hardware designs with one, two, three or four IME modules, we have found that the design that includes only one HW unit is the one that provides the best trade-off among the speed-up, coding performance, power consumption and FPGA area usage. This is mainly due to the intrinsic characteristics of the HEVC encoder application. Although the parallel version of HEVC based on slices has a good scalability in terms of speed-up, the software only makes use of all hardware units simultaneously a few times. This only occurs at the beginning of each frame when all threads are synchronised and use all the available hardware units at the same time to compute the first CTU. However, as the encoding process continues with the remaining CTUs in the frame, the characteristics of the HEVC encoder and the video sequence content (some CTUs are more complex than others) cause the threads to become desynchronised. Consequently, most of the time, only one IME hardware unit is required at a time (there is no hardware concurrency).

Finally, the hardware IME module could theoretically achieve speeds of 15,152 CTU/s (29.7 fps for a 1080p video sequence). However, the limitations and issues described above cause the total encoding time to increase drastically, resulting in a maximum speed of 2,907 CTU/s (8.5 fps).

V. CONCLUSION

In this work, we have presented a full SW/HW integrated version of the HEVC encoder software. The main building blocks of the proposal are based on the HW IME module and the slice-based parallel version of the HEVC encoder, which were both proposed in previous works. In order to perform the HW/SW integration, we have analysed some issues that limit the overall performance of these modules when they work together, such as the DMA transfer operations, the software overhead provided by the OpenCL API, and the concurrent behaviour of the HEVC slice-based encoder. The main results have shown that a) in the HW IME module, using as many HW units as possible does not mean that better performance results will be obtained, mainly due to the HEVC slice-based encoder behaviour where most of the time only one HW unit is required at the same time; b) the DMA data transfer operations are one of the main bottlenecks of the integrated version that significantly reduces the great potential of the IME HW proposal; and c) the use of OpenCL increases the computation overhead even more, especially when the number of threads increases.

After analysing all the impairments found in the integration process, we have selected an integrated version that uses an IME HW with only one HW units and a slice-based SW encoder with no more than eight threads (four threads suggested). The performance behaviour of the integrated version shows that the R/D coding performance is the same as that of the FS SW version (optimal performance), and a speed-up of up to 149.63x is achieved.

There are many tasks that must be completed to appropriately design a more effective HW/SW version of the HEVC encoder. The most important of these are as follows: a) reduce the overhead of the OpenCL API by using a more optimised API, such as the one provided by the FPGA vendor (Xilinx XRT API); b) analyse other schemes to reduce the ratio between data transfer times and CTU computation times so that DMA transfer costs are hidden as much as possible; and c) use different parallel encoding modes, such as chunks of GOPs, or use multiple instances of the encoder to process different video sequences simultaneously so that multiple IME hardware units can be exploited to their fullest potential.

REFERENCES

[1] B. Bross, W.J. Han, J.R. Ohm, G.J. Sullivan, Y-K Wang, and T. Wiegand. High Efficiency Video Coding (HEVC) Text Specification Draft 10. *Document JCTVC-L1003 of JCT-VC, Geneva*, January 2013.

[2] ITU-T and 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.

[3] G.J. Sullivan, J.R. Ohm, W.J. Han, and T. Wiegand. Overview of the high efficiency video coding (HEVC) standard. *Circuits and systems for Video Technology, IEEE Transactions on*, 22(12):1648 –1667, December 2012.

[4] F. Bossen, B. Bross, K. Suhring, and D. Flynn. HEVC Complexity and Implementation Analysis. *Circuits and Systems for Video Technology, IEEE Transactions on*, 22(12):1685–1696, 2012.

[5] Ahmed Medhat, Ahmed Shalaby, Mohammed S Sayed, and Maha Elsabrouty. A highly parallel SAD architecture for motion estimation in HEVC encoder. In *IEEE Asia Pacific Conference on Circuits and Systems (APCCAS'14)*, pages 280–283, Ishigaki, November 2014.

[6] J. Byun, Y. Jung, and J. Kim. Design of integer motion estimator of HEVC for asymmetric motion-partitioning mode and 4K-UHD. *Electronics Letters*, 49(18):1142–1143, 2013.

[7] Xu Yuan, Liu Jinsong, Gong Liwei, Zhang Zhi, and Robert K.F. Teng. A high performance VLSI architecture for integer motion estimation in HEVC. In *IEEE 10th International Conference on ASIC (ASICON'13)*, pages 1–4, Shenzhen, October 2013.

[8] Thomas D'huys. Reconfigurable data flow engine for HEVC motion estimation. In *IEEE International Conference on Image Processing (ICIP'14)*, pages 1223–1227, Paris, October 2014.

[9] Purnachand Nalluri, Luis Nero Alves, and Antonio Navarro. High speed SAD architectures for variable block size motion estimation in HEVC video coding. In *IEEE International Conference on Image Processing (ICIP'14)*, pages 1233–1237, Paris, October 2014.

[10] Belal Mohamed, Ahmed Shalaby, and Mohammed S Sayed. High-level synthesis hardware accelerators of integer-pixel motion estimation

of HEVC on SoC FPGA platform. In *2nd Europe - Middle East - North African Regional Conference of the International Telecommunications Society (ITS): Leveraging Technologies For Growth*, February 2019.

[11] S. Gogoi and R. A Peesapati. A hybrid hardware oriented motion estimation algorithm for HEVC/H.265. *Journal of Real Time Image Processing*, 18, January 2021.

[12] Xilinx. standalone-BSP Board Support Package. (online last access 1/05/2020, 2020. https://www.xilinx.com/support/documentation-navigation\-/design-hubs/dh0041-zc7000-video-and-imaging-kit-hub.html.

[13] Randa Khemiri, Hassan Kibeya, Hassen Loukil, Fatma Ezahra Sayadi, Mohamed Atri, and Nouri Masmoudi. Real-time motion estimation diamond search algorithm for the new high efficiency video coding on FPGA. *Analog Integrated Circuits and Signal Processing*, 94(2):259–276, Aug 2018.

[14] R. Haddar, A. Chaari, H. Kibeya, M. A. Ben Ayed, and N. Masmoudi. FPGA-based implementation of TZsearch algorithm for H.265/HEVC standard. In *2017 18th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering (STA)*, pages 605–610, Dec 2017.

[15] Vladimir Afonso, Henrique Maich, Luan Audibert, Bruno Zatt, Marcelo Porto, Luciano Agostini, and Altamiro Susin. Hardware implementation for the HEVC fractional motion estimation targeting real-time and low-energy. *Journal of Integrated Circuits and Systems*, 11(2), December 2020.

[16] E. Alcocer, O. Lopez-Granado, M. P. Malumbres, and R. Gutierrez. Evaluation of an HEVC hardware IME module using a SoC platform. In *2016 Conference on Design of Circuits and Integrated Systems (DCIS)*, pages 1–6, Nov 2016.

[17] E. Alcocer, R. Gutierrez, O. Lopez-Granado, and M.P. Malumbres. Design and implementation of an efficient hardware integer motion estimator for an HEVC video encoder. *Journal of Real-Time Image Processing*, pages 1–11, 2016.

[18] Lam Duc Khai. A fast and efficient data reuse scheme for hevc integer motion estimation hardware architecture. *Journal of Information and Telecommunication*, 9(1):73–90, 2025.

[19] W. Ahmad, H. Mahdavi, and I Hamzaoglu. An efficient versatile video coding motion estimation hardware. *Journal of Real Time Image Processing*, 21(25), January 2024.

[20] Jun Zhang, Yu Zhang, and Hao Zhang. An efficient hardware architecture of integer motion estimation based on early termination and data reuse for versatile video coding. *Expert Systems with Applications*, 242:122706, 2024.

[21] Shushi Chen, Leilei Huang, Zhao Zan, Xiaoyang Zeng, and Yibo Fan. An interpolation-free fractional motion estimation algorithm and hardware implementation for VVC. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 33(2):395–407, 2025.

[22] Jingping Hong, Zhihong Dong, Zetao Kang Mengxin Pang and, and Peng Cao. Hardware implementation of iterative method for enhanced affine motion estimation in versatile video coding. *Journal of Real Time image Processing*, 22(18), 2025.

[23] Mateus Melo, Gustavo Smaniotto, Henrique Maich, Luciano Agostini, Bruno Zatt, Leomar Rosa, and Marcelo Porto. A parallel motion estimation solution for heterogeneous system on chip. In *2016 29th Symposium on Integrated Circuits and Systems Design (SBCCI)*, pages 1–6, 2016.

[24] Muhammad Usman Shahid, Ashfaq Ahmed, Maurizio Martina, Guido Masera, and Enrico Magli. Parallel H.264/AVC fast rate-distortion optimized motion estimation by using a graphics processing unit and dedicated hardware. *IEEE Transactions on Circuits and Systems for Video Technology*, 25(4):701–715, 2015.

[25] Jinglin Zhang, Jean-Francois Nezan, and Jean-Gabriel Cousin. Implementation of motion estimation based on heterogeneous parallel computing system with OpenCL. In *2012 IEEE 14th International Conference on High Performance Computing and Communication*, pages 41–45, 2012.

[26] M. de Castro, R.R. Osorio, D.L. Vilariño, A. Gonzalez-Escribano, and Llanos D. Implementation of a motion estimation algorithm for intel FPGAs using OpenCL. *Journal of Supercomputing*, 79, June 2023.

[27] S. Radicke, J. Hahn, C. Grecos, and Qi Wang. A multi-threaded full-feature HEVC encoder based on wavefront parallel processing. In *2014 International Conference on Signal Processing and Multimedia Applications (SIGMAP)*, pages 90–98, Aug 2014.

[28] E. Ryu, J. Nam, S. Lee, H. Jo, and D.Sim. Sample adaptive offset parallelism in HEVC. *Multimedia and Ubiquitous Engineering. Lecture Notes in Electrical Engineering*, 240, 2013.

[29] H. Migallón, V. Galiano, P. Piñol, O. López-Granado, and M. P. Malumbres. Distributed Memory Parallel Approaches for HEVC Encoder. *The Journal of Supercomputing*, pages 1–12, 2016.

[30] P. Piñol, H. Migallón, O. López-Granado, and M.P. Malumbres. Slice-based parallel approach for HEVC encoder. *Journal of Supercomputing*, 71:1882–1892, 2015.

[31] Héctor Migallón, Otoniel López-Granado, Vicente Galiano, Pablo Piñol, and Manuel P. Malumbres. *Shared Memory Tile-Based vs Hybrid Memory GOP-Based Parallel Algorithms for HEVC Encoder*, pages 521–528. Springer International Publishing, Cham, 2016.

[32] Gabriel Cebrián-Márquez, José Luis Martínez, and Pedro Cuenca. Inter and intra pre-analysis algorithm for HEVC. *Journal of Supercomputing*, 73:414–432, 2019.

[33] Gabriel Cebrián-Márquez, José Luis Martínez, and Pedro Cuenca. Adaptive inter CU partitioning based on a look-ahead stage for HEVC. *Signal Processing: Image Communication*, 76:97–108, 2019.

[34] W. Chen and X. Wang. Fast entropy-based cabac rate estimation for mode decision in HEVC. *SpringerPlus*, 756, 2016.

[35] G. Correa, P. A. Assuncao, L. V. Agostini, and L. A. da Silva Cruz. Fast HEVC encoding decisions using data mining. *IEEE Transactions on Circuits and Systems for Video Technology*, 25(4):660–673, April 2015.

[36] Rjoji Tsuchiyama, Takashi Nakamura, Takuro Iizuka, Akihiro Asahara, and Satoshi Miki. *The OpenCL Programming Book*. Fixstars Corporation, 2009.

[37] I. Kim, J. Min, T. Lee, W. Han, and J. Park. Block partitioning structure in the HEVC standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(12):1697–1706, Dec 2012.

[38] P. Piñol, H. Migallón, O. López-Granado, and M. P. Malumbres. Slice-based parallel approach for hevc encoder. *The Journal of Supercomputing*, 71(5):1882–1892, May 2015.

[39] Fraunhofer-HHI. HEVC Reference Software (HM-16.3), 2015. available at: http://hevc.hhi.fraunhofer.de/svn/ svn_HEVCSoftware/tags/HM-16.3/.

OTONIEL LÓPEZ-GRANADO received his M.S in Computer Science from the University of Alicante (Spain) in 1996. Between 1997-2003 he worked as programmer analyst in an important industrial informatics firm. In 2003 he joined to the Computer Engineering Departament at Miguel Hernandez University (UMH), Spain, as an assistant professor. Then, he received the phD degree in Computer Science in 2010. In 2020 he was promoted to Full professor. From 2015 to 2024 he lead the GATCOM research group (atc.umh.es) at Miguel Hernandez University. He is author of more than 100 conference and journal publications. His research and teaching activities are related to multimedia networking (audio/video coding and network delivery).

HÉCTOR MIGALLÓN was born in Alicante, Spain in 1972. He received the M.S. degree in Physics in 1995 and the M.S. degree in Electronic Engineering in 2011 from the University of Valencia, Spain, and the Ph.D. degree in computer science from Miguel Hernández University, Spain, in 2005.

He is member of the "Architecture and Computer Technology" research group at the Miguel Hernández University and the "High Performance Computing and Parallelism" research group at the University of Alicante. His main research interests include parallel algorithms for solving linear and nonlinear systems, parallel algorithms for image and video processing, parallel heuristic optimization algorithms and parallel high level interfaces for heterogeneous platforms.

ESTEFANIA ALCOCER was born in Bigastro, Spain, in 1986. She received her M.S. degree in telecommunication engineering in 2010 from Miguel Hernandez University, Elche, Spain, and she joined the GATCOM research group as Ph.D. student in 2012. Since 2013 to 2017 she was assistant professor in the Department of Physics and Computer Architecture at Miguel Hernandez University. In 2017 she received the phD degree in Telecommunications from Miguel Hernandez University. Currently, she act as Teacher in a Secondary school at Torrevieja. Her current research activities are related to image processing, the design of FPGA-based systems and video coding.

ROBERTO GUTIERREZ MAZON was born in Orihuela, Spain, in 1977. He received his M.Sc. degree on Telecommunication Engineering in 2003, and the PhD degree in Electronic Engineering in 2011, both from the Universidad Politecnica de Valencia, Spain. He is Associate Professor in the Department of Communication Engineering at Universidad Miguel Hernandez, Elche since 2019. His current research interests include the design of FPGA-based systems, computer arithmetic, VLSI signal processing and digital communications.

GLENN VAN WALLENDAEL received the M.Sc. degree in computer science engineering from Ghent University, Belgium, in 2008, and the Ph.D. degree from IDLab, Ghent University, with the financial support of the Research Foundation – Flanders (FWO). Since 2019, he has been working as a Professor at Ghent University. His main topics of interest are the efficient representation and compression of visual information, including 360 degree video, light field, virtual reality, and the different operations on these modalities, such as (scalable) compression, transcoding, encryption, watermarking, personalized delivery, and quality estimation.

MANUEL P. MALUMBRES received his B.S. in Computer Science from the University of Oviedo (Spain) in 1986, and the M.S. and Ph.D. degrees in Computer Science from Technical University of Valencia (UPV), at 1991 and 1996 respectively. He is author of more than 200 conference and journal publications and several networking books for undergraduate CS courses. Currently, his research and teaching activities are related to multimedia networking (image/video coding and network delivery), wireless network technologies (MANETs, VANETs and WSNs), and hardware acceleration schemes for multimedia applications (multi/manythreads, GPUs, FPGAs).

TABLE 6. Evaluation results of the SW/HW integrated version with one IME hardware unit with the CTC video sequences.

Class	Sequence	Measure	Threads						
			1	2	4	8	12	16	20
A+	NebutaFestival	Time(s)	15824.4	8127.6	4261.7	2261.7	1654.3	1386.5	1172.11
		Speed-up	-	1.95	3.71	7.00	9.57	11.41	13.50
		PSNR(db)	29.45	29.45	29.44	29.43	29.42	29.41	29.40
		Bitrate(Mb)	19.49	19.56	19.64	19.76	19.87	19.98	20.07
	SteamLocomotive	Time(s.)	10312.5	5450.1	2989.4	1615.4	1191.0	1006.9	842.9
		Speed-up	-	1.89	3.45	6.38	8.66	10.24	12.23
		PSNR(db)	38.68	38.66	38.64	38.63	38.61	38.60	38.59
		Bitrate(Mb)	2.03	2.06	2.09	2.15	2.21	2.26	2.29
A	Traffic	Time(s)	4690.7	2431.7	1342.6	727.3	541.7	450.7	378.2
		Speed-up	-	1.93	3.49	6.45	8.66	10.41	12.40
		PSNR(db)	36.54	36.53	36.52	36.50	36.49	36.48	36.48
		Bitrate(Mb)	1.58	1.60	1.63	1.65	1.67	1.69	1.71
	PeopleOnStreet	Time(s.)	6518.8	3379.3	1909.6	1007.7	749.9	625.9	522.9
		Speed-up	-	1.93	3.49	6.45	8.66	10.41	12.40
		PSNR(db)	34.14	34.12	34.11	34.10	34.08	34.08	34.07
		Bitrate(Mb)	5.06	5.14	5.20	5.26	5.30	5.35	5.39
B	BQTerrace	Time(s)	9785.1	5199.9	2778.4	1493.3	1112.2	931.6	800.4
		Speed-up	-	1.88	3.52	6.55	8.80	10.50	12.23
		PSNR(db)	33.84	33.83	33.81	33.80	33.79	33.78	33.77
		Bitrate(Mb)	3.59	3.63	3.68	3.76	3.84	3.91	3.96
	BasketballDrive	Time(s)	9478.6	5085.8	2885.9	1564.5	1180.6	960.0	826.8
		Speed-up	-	1.86	3.28	6.06	8.03	9.87	11.46
		PSNR(db)	35.62	35.60	35.58	35.57	35.55	35.53	35.53
		Bitrate(Mb)	3.74	3.81	3.88	4.01	4.13	4.24	4.30
	Cactus	Time(s)	8468.2	4418.4	2386.6	1342.0	1019.7	834.2	729.8
		Speed-up	-	1.92	3.55	6.31	8.30	10.15	11.60
		PSNR(db)	34.93	34.91	34.88	34.86	34.85	34.83	34.83
		Bitrate(Mb)	3.57	3.60	3.67	3.77	3.86	3.94	3.98
	Kimono	Time(s)	4268.0	2303.8	1231.8	661.3	490.1	408.7	350.9
		Speed-up	-	1.85	3.46	6.45	8.71	10.44	12.16
		PSNR(db)	37.39	37.35	37.32	37.28	37.26	37.23	37.23
		Bitrate(Mb)	1.29	1.31	1.34	1.38	1.42	1.46	1.48
	ParkScene	Time(s)	4089.3	2138.4	1159.2	631.5	468.4	394.5	336.7
		Speed-up	-	1.91	3.53	6.47	8.73	10.36	12.14
		PSNR(db)	34.88	34.86	34.84	34.83	34.81	34.80	34.79
		Bitrate(Mb)	1.79	1.81	1.83	1.87	1.91	1.94	1.96
C	BQMall	Time(s)	2143.7	1138.9	687.9	380.9	290.6	253.8	236.1
		Speed-up	-	1.88	3.12	5.63	7.38	8.45	9.08
		PSNR(db)	34.98	34.93	34.90	34.86	34.85	34.85	34.85
		Bitrate(Mb)	1.18	1.22	1.27	1.35	1.40	1.43	1.46
	BasketballDrill	Time(s)	1899.1	1019.2	573.3	319.0	247.6	219.2	204.5
		Speed-up	-	1.86	3.31	5.95	7.67	8.66	9.28
		PSNR(db)	34.36	34.35	34.33	34.30	34.30	34.29	34.29
		Bitrate(Mb)	1.09	1.11	1.15	1.21	1.25	1.28	1.30
	PartyScene	Time(s)	2001.7	1112.7	616.8	334.8	259.8	230.2	217.2
		Speed-up	-	1.80	3.25	5.98	7.70	8.69	9.21
		PSNR(db)	31.67	31.65	31.62	31.59	31.59	31.58	31.58
		Bitrate(Mb)	1.97	2.00	2.04	2.09	2.13	2.15	2.18
	RaceHorsesC	Time(s)	1408.4	780.3	421.4	227.6	175.8	152.9	144.4
		Speed-up	-	1.80	3.34	6.19	8.01	9.21	9.75
		PSNR(db)	32.90	32.88	32.86	32.82	32.82	32.81	32.81
		Bitrate(Mb)	1.17	1.19	1.22	1.27	1.29	1.31	1.33
	BasketDrillText	Time(s)	1912.9	1025.0	576.2	328.1	252.7	223.1	206.6
		Speed-up	-	1.87	3.32	5.83	7.57	8.57	9.26
		PSNR(db)	34.28	34.25	34.24	34.21	34.20	34.21	34.20
		Bitrate(Mb)	1.20	1.23	1.26	1.33	1.37	1.39	1.42

TABLE 7. Computing time, speed-up, PSNR, and bitrate for video sequences (classes D to F) using HW2 IME module and RA configuration (QP=32).

Class	Sequence	Measure	Threads						
			1	2	4	8	12	16	20
D	BQSquare	Time(s)	507.3	295.0	164.6	110.5	92.1	76.1	84.4
		Speed-up	-	1.72	3.08	4.59	5.50	6.67	6.01
		PSNR(db)	32.00	31.96	31.93	31.90	31.89	31.87	31.87
		Bitrate(Mb)	0.44	0.45	0.48	0.51	0.54	0.57	0.59
	BasketballPass	Time(s)	516.8	283.8	175.4	117.4	98.2	80.9	87.2
		Speed-up	-	1.82	2.95	4.40	5.26	6.38	5.93
		PSNR(db)	33.54	33.51	33.47	33.46	33.45	33.44	33.44
		Bitrate(Mb)	0.47	0.49	0.52	0.55	0.58	0.61	0.63
	BlowingBubbles	Time(s)	463.9	257.8	148.8	100.5	85.8	70.1	77.3
		Speed-up	-	1.80	3.12	4.62	5.40	6.62	6.00
		PSNR(db)	31.73	31.68	31.65	31.64	31.63	31.62	31.62
		Bitrate(Mb)	0.48	0.49	0.51	0.54	0.56	0.58	0.60
	RaceHorses	Time(s)	344.2	206.0	113.4	75.0	63.7	50.2	54.5
		Speed-up	-	1.67	3.03	4.59	5.40	6.85	6.31
		PSNR(db)	32.27	32.24	32.19	32.17	32.15	32.15	32.15
		Bitrate(Mb)	0.35	0.36	0.38	0.39	0.41	0.43	0.44
E	KristenSara	Time(s)	3926.5	2079.3	1149.8	619.8	447.3	387.0	342.3
		Speed-up	-	1.89	3.41	6.34	8.78	10.14	11.47
		PSNR(db)	39.27	39.24	39.22	39.18	39.15	39.15	39.15
		Bitrate(Mb)	0.62	0.64	0.67	0.72	0.77	0.80	0.83
	Johnny	Time(s)	3742.8	2032.0	1087.2	580.6	430.2	366.4	328.2
		Speed-up	-	1.84	3.44	6.45	8.70	10.21	11.40
		PSNR(db)	39.59	39.58	39.56	39.52	39.50	39.49	39.50
		Bitrate(Mb)	0.43	0.45	0.47	0.52	0.57	0.60	0.63
	FourPeople	Time(s)	3840.0	2108.4	1193.1	661.1	488.3	412.4	360.4
		Speed-up	-	1.82	3.22	5.81	7.86	9.31	10.65
		PSNR(db)	38.20	38.17	38.15	38.12	38.09	38.09	38.08
		Bitrate(M)	0.86	0.88	0.91	0.96	1.00	1.03	1.06
	SlideEditing	Time(s)	1778.1	980.5	528.5	286.7	211.2	182.2	161.6
		Speed-up	-	1.81	3.36	6.20	8.42	9.761	11.00
		PSNR(db)	38.38	38.37	38.36	38.32	38.30	38.28	38.28
		Bitrate(M)	0.95	0.96	0.98	1.00	1.02	1.04	1.06
F	ChinaSpeed	Time(s)	3877.6	2094.4	1207.6	673.5	515.1	426.0	373.6
		Speed-up	-	1.85	3.21	5.76	7.53	9.10	10.38
		PSNR(db)	34.91	34.91	34.89	34.87	34.86	34.85	34.84
		Bitrate(Mb)	2.68	2.70	2.74	2.80	2.85	2.89	2.92
	SlideShow	Time(s)	3379.8	1836.6	1009.1	558.8	413.7	355.5	321.9
		Speed-up	-	1.84	3.35	6.05	8.17	9.51	10.50
		PSNR(db)	42.27	42.27	42.28	42.26	42.24	42.22	42.22
		Bitrate(Mb)	0.85	0.86	0.89	0.95	0.99	1.03	1.06

TABLE 8. Performance evaluation of different ME algorithms and our IME HW module.

ME	Threads	Time (*s*)	ME/CTU (*ms*)	CTU_SpUp
FS	1	8089.40	1026.647	-
	2	4268.61	1035.193	-
	4	2283.14	1086.514	-
	8	1206.03	1114.553	-
	12	859.40	1173.598	-
	16	726.88	1322.881	-
	20	634.00	1411.241	-
Selective	1	626.22	50.244	20x
	2	329.27	51.177	20x
	4	182.00	54.201	20x
	8	106.66	55.975	20x
	12	78.73	59.201	20x
	16	67.30	66.398	20x
	20	58.32	70.876	20x
eDiamond	1	394.63	20.451	50x
	2	206.27	20.368	51x
	4	111.96	21.296	51x
	8	62.17	21.793	51x
	12	45.44	22.899	51x
	16	39.16	25.815	51x
	20	34.83	27.410	51x
Diamond	1	276.99	5.389	190x
	2	145.67	5.552	187x
	4	79.23	5.902	184x
	8	44.78	6.066	184x
	12	33.25	6.401	183x
	16	28.79	7.223	183x
	20	25.78	7.717	183x
FS-HW1	1	252.45	1.511	679x
	2	134.77	1.614	641x
	4	74.28	1.780	604x
	8	42.93	2.095	532x
	12	32.96	2.508	468x
	16	28.28	3.078	430x
	20	25.11	3.696	382x

TABLE 9. Evaluation of SW FS version and the HW/SW proposal with four threads.

Video Sequence	SW/HW FS			SW FS			Speed-up	PSNR	Bitrate
	Total Time (s)	PSNR (dB)	Bitrate (Mb)	Total Time (s)	PSNR (dB)	Bitrate (Mb)			
NebutaFestival	4261.77	29.44	19.643	400876.0	29.43	19.442	94.06x	-0,02%	1.03%
Traffic	1342.69	36.52	1.632	183217.0	36.55	1.584	136.45x	0.09%	3.04%
PeopleOnStreet	1909.64	34.11	5.204	201067.7	34.14	5.119	105.29x	0.10%	1.67%
BQTerrace	2778.43	33.81	3.687	365374.1	33.83	3.546	131.50x	0.06%	3.96%
BasketballDrive	2885.97	35.58	3.885	313520.6	35.63	3.583	108.64x	0.14%	8.42%
Cactus	2386.68	34.88	3.678	308749.8	34.94	3.579	129.36x	0.16%	2.78%
Kimono	1231.86	37.32	1.342	147229.5	37.40	1.312	119.52x	0.22%	2.26%
ParkScene	1159.24	34.84	1.836	147030.7	34.88	1.805	126.83x	0.11%	1.73%
BasketballDrill	573.33	34.33	1.154	60549.6	34.39	1.068	105.61x	0.17%	8.04%
BasketDrillText	576.24	34.24	1.267	60776.0	34.31	1.181	105.47x	0.19%	7.29%
BQSquare	164.64	31.92	0.480	16701.3	31.99	0.434	101.44x	0.19%	10.73%
BasketballPass	175.43	33.47	0.529	14737.6	33.56	0.471	84.01x	0.28%	12.21%
BlowingBubbles	148.88	31.65	0.514	14211.9	31.75	0.476	95.46x	0.31%	8.00%
RaceHorses	113.48	32.19	0.380	9251.5	32.30	0.354	81.52x	0.34%	7.23%
KristenSara	1149.83	39.22	0.672	160404.3	39.27	0.623	139.50x	0.15%	7.83%
Johnny	1087.28	39.56	0.477	159718.1	39.60	0.439	146.90x	0.11%	8.63%
FourPeople	1193.12	38.15	0.916	161274.3	38.20	0.868	135.17x	0.13%	5.49%
SlideEditing	528.56	38.36	0.980	79089.7	38.35	0.914	149.63x	-0.02%	7.27%
ChinaSpeed	1207.66	34.89	2.746	122858.6	34.92	2.683	101.73x	0.08%	2.35%
SlideShow	1009.14	42.28	0.893	134642.1	42.45	0.792	133.42x	0.40%	12.76%

TABLE 10. Delay chain of SW & HW operations to compute SAD&MV of a CTU in μs.

Device	OpenCL		IME HW Module (249 MHz)			OpenCL	
	OCL_Schedule	DMA (CTU&SAD)	HBM to IP	ME	IP to HBM	DMA (SADS&MVs)	Other
Host	7.5	37				31	37.5
FPGA			162.6	66	2.4		