Scalable Heterogeneous Accelerating Structure for the HEVC Temporal Prediction

João Luís Henriques Afonso  
Instituto Superior Técnico, University of Lisbon, Portugal  
joao.luis.afonso@tecnico.ulisboa.pt

Abstract—Despite offering significant gains in terms of the resulting encoding efficiency, the state-of-the-art High Efficiency Video Coding (HEVC) standard is characterized by a significant computational complexity, making it hardly implemented in real-time by nowadays General Purpose Processors (GPPs). This is mainly due to the Motion Estimation (ME) part of its temporal prediction module - a very demanding and repetitive operation that takes a significant portion of the processing time. To circumvent such constraint, a scalable heterogeneous accelerating structure is presented, consisting of a set of Processing Elements (PEs) able to process several blocks of the current frame (the CTUs) in parallel. These PEs are composed by highly efficient accelerators, dedicated to the most demanding ME parts: integer search and the sub-pixel refinement operations. Moreover, the conceived PEs are fully programmable, thus supporting the implementation of most state-of-the-art ME algorithms and search strategies. Furthermore, a highly efficient distributed memory mechanism was also devised, in order to satisfy the demanding data transfers of the pixel data from the main encoder memory to each PE, without constraining the scalability of the system. To test and evaluate the proposed scalable acceleration system, an aggregate of 16 PEs was prototyped in a Virtex-7 FPGA, connected to a high-end desktop GPP via PCIe. The obtained results showed that the distributed memory mechanism and the devised accelerators are able to process the input video stream in real-time (30 fps), up to the resolution of 2160p (4K-UHD).

I. INTRODUCTION

In 2013, a new video coding standard was proposed, named HEVC [1], which aims to reduce the size of the encoded video bitstream, at around 50%, in comparison to the previous standard H.264/AVC, while preserving the same perceptive visual quality. However, this comes with a much higher computational cost for encoding and decoding.

Therefore, the work presented in this dissertation aims to accelerate the most complex step of a HEVC encoder: the interframe prediction. This part of the encoder may take up to 60% of the encoding time [2], due to the exhaustive nature of ME procedure. This is due to the ME requiring a significant number of comparisons between these blocks and the reference frames. This, added to the fact that ultra-high definition formats have a larger area, with many more blocks, results in a very exhaustive ME procedure. Fortunately, since these comparisons are very repetitive, they are a great candidate for being accelerated with dedicated parallel architectures. Hence, the work developed for this dissertation aimed to accelerate the interframe prediction by providing the means to massively parallelize the ME procedure. It consists of a distributed scheme, containing a scalable number Acceleration Units ( AU s), each one able to process different Coding Tree Units ( CTUs ), and respective sub-blocks. Inside each one of these AU s, a set of dedicated accelerators is used to accelerate the most demanding parts of the ME procedure, while a programmable controller is used to control the steps of the ME.

In addition, a fast and efficient distributed memory mechanism was developed, able to provide the pixel data to a scalable number of AU s and accelerators, simultaneously.

The proposed distributed prediction scheme was implemented in a Virtex-7 FPGA, which allowed 16 PEs to be implemented in parallel. These PEs correspond to a materialization of the AU s. Each one contains two specialized accelerators for the ME, also proposed on this dissertation: one for integer search, the other for sub-pixel refinement. In addition, a local GPP is used to control the various steps executed by the accelerators. By manipulating the program on these GPPs, multiple ME algorithms and strategies can be implemented.

II. TEMPORAL PREDICTED VIDEO ENCODING

According to the HEVC encoder, each picture is split into block-shaped regions of various sizes, the CTUs. For most pictures, interpicture temporal prediction is typically employed for encoding, due to its better efficiency. This process involves selecting, for each block, the reference picture and Motion Vector ( MV ) that better match the block being encoded ( see Fig. 1 ). The operation of this process, denoted as Motion Estimation (ME), allows the encoder to compress the pictures based on how the different partitions move from frame to frame, thus reducing the amount of information required to encode it. According to [2], two of the steps used by the Motion Estimation ( ME ) module of the HEVC interprediction, Sum of Absolute Differences ( SAD ) and subpixel interpolation, take 40% and 20% of the total encoding time, respectively.

A. Motion Estimation

The Motion Estimation ( ME ) analyses the displacement of each image partition in the neighbouring frames and encodes this information using MVs ( see Fig. 1 ). When the frame partitioning and the corresponding MVs are very precise, the residuals becomes very small, thus reducing the total amount of information in the bitstream. However, a high precision analysis usually requires more computations, thus a balance
Fig. 1. Motion vector that describes the displacement of a certain block.

Fig. 2. Three step search algorithm. Fig. 3. Diamond search algorithm.

between the attained bitrate reduction and the computational power required to process it needs to be considered.

To determine the best MV for a certain block, a block matching ME algorithm is usually employed. They involve comparing each small partition of the current frame to the corresponding block and adjacent neighbours in one or more nearby reference frames, inside a search window. The position with the lowest cost is then chosen for the MV. For faster encoders, the cost is usually the SAD between the luma samples of both frames.

B. Integer Search Algorithms

The optimal ME result is obtained with the full-search block-matching algorithm, which exhaustively compares the current block with all the possible positions inside the reference frame search window. This algorithm guarantees that the best match is always found but, due to the very high resolutions of nowadays video formats and complexity of the standards, has become impractical.

Therefore, most application domains have adopted other alternatives based on fast block-matching algorithms. These algorithms only pick a small number of positions on each step, and use the best result for the starting point of the next iteration. After a certain number of iterations, a minimum of the cost function is achieved.

Two of the first fast search algorithms that have been proposed are the Three Step Search (TSS) [3] and Diamond Search (DS) [4] (see figures 2 and 3). More recently, other algorithms have been proposed, such as the Enhanced Predictive Zonal Search (EPZS) [5] and the Test Zone Search (TZS) [6], that achieve better results for high resolution formats at a cost of a higher algorithmic complexity.

C. Sub-Pixel Refinement

Sub-pixel MV refinement is employed after the integer block matching ME algorithm, to get a better estimation of the MV. It usually takes the two steps illustrated in Fig. 4. In the first step, the block is compared to eight half-pixel precision locations around the best integer-precision MV. After this, a quarter-pixel precision comparison takes place, by using the best half-pixel location as a starting point.

D. Block Organization for Temporal Prediction

To improve the block partitioning efficiency and achieve a better bit rate reduction, the HEVC standard introduced the CTU structure (see Fig. 6). Each CTU consists of one luma and two chroma components, each one represented by a Coding Tree Block (CTB), and the associated syntax (Fig. 5). Unlike the macroblock structure from previous standards, the CTUs can have a size up to $64 \times 64$.

Each CTU and CTB can be further partitioned into Coding Units (CUs) and Coding Blocks (CBs), respectively, by following a quad-tree structure (see figures 6 and 7).

In addition, each CU is associated with a group of Prediction Units (PUs), which can be generated by partitioning the CU into 8 possible prediction modes (see Fig. 8). Usually, the luma Prediction Blocks (PBs) are the blocks that are compared to the reference frames, using the considered motion search algorithm, in order to find the best MV. When encoding a video sequence, the encoder can chose which CU sizes and PU modes to analyse.

To further improve the encoding efficiency, some HEVC features impose dependencies, by using the MVs information from neighbour PUs to improve the encoding efficiency of each PU. If these dependencies are to be respected, each PU can only be processed after the neighbour left, top-left, top-middle and top-right PUs had been processed. Fortunately, HEVC specifies two new parallelization approaches to accelerate the encoding procedure, while allowing information to be inherited from neighbour blocks.

The first one consists on the usage of Tiles. This option allows the frame to be partitioned into a number of rectangular regions, usually with the same size, where intra and inter-prediction can be independently performed. Another feature introduced in HEVC is Wavefront Parallel Processing (WPP). With WPP, each frame is divided into horizontal rows of CTUs. Each of these rows can be encoded in parallel, provided that the row above is at least two CTUs ahead.

As it will be described in the following sections, the architecture presented in this thesis aims to be compatible with several parallelization methods. It is important to note
that, in order to support a greater level of parallelism, these dependencies can be disabled, albeit at a cost of a greater bitrate.

III. STATE OF THE ART

In [7], an improvement of the WPP scheme is presented, called Overlapped Wave-front (OWF). Two other approaches, Global Parallel Method (GPM) and Local Parallel Method (LPM), described by [8], aim to further increase the parallelism of HEVC, thus making it more suitable to be implemented in many-core architectures.

In [9], [10], it is presented an architecture design for the HEVC ME, which processes a large number of pixels in parallel, albeit consuming a significant number of resources. In [11] several configurations of a ME architecture are analysed, that uses several different processing modules, each specialized in a different PU size. These two contributions consider the both the integer and fractional search, but are not adapted for being implemented in multi-core scalable architectures.

Other contributions only focus on the integer search or the interpolation. Several configurations of a simple SAD pipeline structure are studied by [12], regarding performance versus energy consumption. In [13] an architecture specialized on the HEVC SAD task, for blocks of various sizes, is presented. Other contributions, as [14]–[16], focus on the interpolation procedure, which is considerably more demanding for HEVC than for previous standards, thus requiring new specialized accelerating hardware. These architectures require a significant amount of memory in order to store the interpolated pixels. Contrary to these, the sub-pixel refinement accelerator herein presented is able to compare the pixels right after being interpolated, thus requiring much less memory and allowing larger areas to be interpolated.

IV. PROPOSED TEMPORAL PREDICTION PARALLELIZATION MODEL

A new model for the parallelization of the HEVC temporal prediction is herein presented. This system aims to benefit from various levels of parallelism, while being compatible with different rules - ranging from less to more strict - regarding the dependencies between the various blocks, and also aims to be compatible with several ME algorithms and strategies. The proposed model is depicted in Fig. 9, at its higher abstraction level. It contains multiple Acceleration Units (AUs), which are optimized to handle the interframe prediction of a large number of CTUs in parallel. From the encoder point of view, the CTUs are distributed amongst the AUs when they are available to process new CTUs. Then, after being processed, the motion data of the various sub-blocks of each CTU is returned to the encoder, to be used by the other modules.

The pixel data communication signal, between the main encoder and the accelerator, is the most demanding in terms of required bandwidth, since it needs to transfer all the CTU and search window pixels for the AUs. It may require transfer rates in the order of GB/s for real time encoding of high-definition formats at 30fps, as it is referred in Table I (for one reference frame only).

<table>
<thead>
<tr>
<th>Resolution</th>
<th>Search window size</th>
</tr>
</thead>
<tbody>
<tr>
<td>±32</td>
<td>±64</td>
</tr>
<tr>
<td>720p</td>
<td>141</td>
</tr>
<tr>
<td>1080p</td>
<td>299</td>
</tr>
<tr>
<td>1440p</td>
<td>539</td>
</tr>
<tr>
<td>2160p</td>
<td>1195</td>
</tr>
</tbody>
</table>

Therefore, a dedicated and rather efficient communication infrastructure is required to transfer these pixels to the AUs. The pixel data is then stored in a set of dedicated memories, inside each acceleration unit (pixel memories), in order to be locally accessed by the accelerators when processing its CUs and PUs.

In order to avoid stalling the AUs while waiting for new pixel data, an implicit double buffering scheme is employed, where one CTU is being processed by the AU while the next one is being (or waiting to be) transferred (see Fig. 10).
Fig. 11. Acceleration Unit sub-modules.

Fig. 12. AU local Pixel Memory, with its reference and current frame pixels modules.

Fig. 13. Current frame address space.

Fig. 14. Reference frame address space.

Fig. 15. Search window, centered.

Fig. 16. Search area reuse for neighbour search windows.

Fig. 17. Partitioning of the reference frame in 32×32 partitions, mapped to physical memory slots.

Accelerators, responsible for managing the most repetitive and demanding steps of the ME procedure (SAD, interpolation, sub-pixel refinement, or others). In order to decouple their operation from the controller, each accelerator has an input and an output FIFO, for the commands and the results, respectively.

Pixel memory, used to store the pixels of the CTU (and corresponding search window) being processed, as well as the CTU being prefetched. This memory has to supply the multiple accelerators with arrays of pixels, while being able to simultaneously receive new data from the Direct Memory Access (DMA) controller.

Controller, responsible for managing the ME of the various sub-blocks (PUs) of the CTU, by sending commands to the accelerators, interpreting the results and communicating with the main encoder.

A. Distributed Memory Organization

All the CTU and respective search windows pixels are temporarily stored in a local pixel memory, so that the accelerators of the AU are able to access it with a very low latency and high throughput (Fig. 11). It is able to supply pixel data to all the accelerators in every clock cycle.

These pixel memories are composed of two main modules which are accessed simultaneously: one for the reference frame pixels, the other for the current frame pixels (see fig. 12). This is very advantageous, since most motion search operations involve comparing one set of current frame pixels to another set of reference frame pixels.

In addition, these memories are accessed by the accelerators using a 2D address space (see Fig. 13 and 14). This approach is better suited for ME procedure, and helps hiding the internal memory organization from the accelerators. Besides this, two more signals are used to select between two CTU slots (one being processed, the other prefetched) and between several search windows.

Three levels of data reusage are adopted by the proposed parallel temporal prediction model:

1) From PU to PU. This is done by transferring the entire search window of the CTU to the pixel memory. This data can then be reused for each of the CTU sub-blocks, including the PUs, thus avoiding new data transfers (see Fig. 15).

2) From having multiple accelerators simultaneously accessing the same pixel memory, which avoids having to duplicate the pixel data.

3) From CTU to CTU. If their search windows share a portion of the reference frame, this data does not have to be transferred twice (see Fig. 16). This provides a bandwidth reduction as high as 71% (between 256×256 sized search windows).
V. PROPOSED TEMPORAL PREDICTION ARCHITECTURE

The architecture herein presented (see Fig. 18) aggregates a scalable number of PEs, which materialize the AUs. Internally, each PE contains: a GPP, responsible for implementing the search algorithm and strategy control; one accelerator for integer search; another accelerator for sub-pixel refinement; and the pixel memories. It also includes some registers, shared with the rest of the encoder, as well as a clock cycle counter, which can be used for debugging. Besides the PEs, there are also other modules that can be used to exchange information between the PEs or with the main HEVC encoder.

The other parts of the HEVC encoder (not included in the architecture) could be either implemented with dedicated hardware or on a high-end host GPP. For this work, the latter option was considered. Therefore, for the communication between the proposed accelerating structure and the host GPP, a PCI Express (PCIe) interconnection was chosen (based on the framework presented by [17]). For this communication, four channels are considered:

- **Memory Mapped Control** - Used by the host GPP to access the memory mapped addresses of the shared memory, mutexes, shared registers (of each PE) and the instruction memory configuration module.
- **Interrupt Interface** - Used by the accelerator to notify the host of certain events.
- **Output Stream** - Buffered stream of data to be sent from the PEs to the host memory (with the ME results).
- **Input Stream** - Buffered stream of data to be sent from the host memory to the PEs (with the pixel data).

The software control layer depicted in Fig. 18 controls the addresses of the main memory where the input/output stream of data is going to be written or read from.

A. Pixel Memory Architecture

Since the PEs of the architecture contains two accelerators, this implementation of the pixel memory needs to have two independent reading ports. One additional writing port is used to receive the new pixel data from the host main memory.

Fig. 19 illustrates the set of the signals that are used to address each reading port. This memory allows both accelerators to receive new pixel data at each cycle, provided that the signals read enable and valid are active by both sides. Two horizontal arrays of 16 pixels are obtained for each accelerator: one with current frame pixel data, the other with reference frame pixel data. This allows the accelerators to benefit from a fine-grained level of parallelism.

Internally, the 2D addresses are mapped to physical addresses of the current and reference pixel data (see Fig. 20). In order to be able to simultaneously provide pixels to the two accelerators, multiple banks are used for the current and reference frame data.

This simultaneous access is guaranteed if two main prerequisites are respected. First, the accelerators should access the pixel memory with the pattern exemplified in Fig. 21. Secondly, both the current and reference frame pixels have to be split among the banks, according to their Y address, as illustrated in Fig. 22.

If a conflict arises (when two accelerators try to access the same bank for reference or current pixels) one of the accelerators has to be put on hold. Fortunately, since the number of banks is large enough (see Fig. 22) and the accelerators access the memory with the described pattern (Fig. 21), it is guaranteed that in a maximum of 3 cycles they will have access to both the current and reference memory banks. After this, as long as the accelerators continue accessing the memory with the same pattern, they will have access to the pixels on every cycle.

The CTUs and the $32 \times 32$ partitions of the reference frames are placed in slots inside of the pixel memory banks, as seen in Fig. 23. In addition, the address X of the reference frame may not be aligned with the borders, which may require an array to be read from two $32 \times 32$ partitions simultaneously. This is due to the nature of ME algorithms, which may require addresses of the search window to be accessed with any (X,Y) offset.

Meanwhile, internally, each physical bank can be implemented by using a dual-port memory. One of the ports is only used for writing (by the Pixel DMA controller), while the other is only for reading (by one of the two accelerators). This allows the pixels to be prefetched while the accelerators are accessing the memory.

In order to translate from the 2D virtual address space into the physical address space, a set of translation modules were developed. For the current frame, the 2D virtual address is directly translated into the physical address, through the concatenation of its values. For the reference frame, the address has to be translated twice, since the corresponding array of pixels may span across two $32 \times 32$ partitions. To allow this, each bank of the reference memory is internally partitioned into two sub-banks (named as columns), which are addressed independently. A Translation Table is used to store the index of the slot where each reference frame partition is stored, thus being able to match the 2D address into the respective slot. They are configured by the pixel DMA controller when a new packet with the data of a $32 \times 32$ partition is received.

B. Integer Search Accelerator

Fig. 24 presents the block diagram of the proposed Integer Search Accelerator. The main purpose of this accelerator is to analyse the cost of a list of MVs, for every step of a given block matching ME algorithm, and return the index of the MV with the lowest cost (calculated with the SAD metric). Both the input and the output streams are communicated through FIFOs, in order to decouple the GPP control from the accelerator.

The command interpreter module is responsible for receiving the commands from the FIFO, interpreting them, forwarding the corresponding parameters to the other modules of the accelerator and signalling them when a new SAD has to be started. There are two commands: one with the configuration of the PU which is to be processes, the other to order the comparison of this PU to one location of a search window.
The second module is responsible for generating the 2D address that indexes the pixel memory. On each cycle, one horizontal array of 16 pixels is read from the current and another from the reference frame, which are forwarded to the next module.

The SAD pipeline module is responsible for computing the SAD with the two arrays of pixels received from the previous module, in pipeline.

Finally, the last module is responsible for accumulating the partial SADs, in order to obtain the full SAD value. After obtaining the SAD, it compares it to the one previously calculated, referent to a different position on the frame, in order to determine which one has the lowest cost. Therefore, when requested by the GPP, the accelerator is able to return to the output FIFO a reference to the best position, along with its corresponding SAD cost.

C. Sub-pixel Refinement Accelerator

The block diagram of the proposed sub-pixel refinement accelerator is given in Fig. 25.

Given a PU and a certain location of the search window, the purpose of this accelerator is to refine it to quarter pixel precision. Two operations are allowed: half-pixel refinement, and quarter-pixel refinement (see Fig. 26). For each one, it compares its SAD with the SAD of the 8 sub-pixel positions around it, and returns the index of the best one, along with the corresponding SAD value.

The architecture of this accelerator intends to be scalable. In order to show this scalability, two pixels from the PU are processed in parallel, in each pipeline stage.

The command interpreter module has the task of receiving the commands from the GPP, interpreting them, forwarding the parameters to the other modules of the accelerator and signalling them when a new sub-pixel refinement is to be started, just like for the integer search accelerator.

The 2D address generator module is responsible for generating the 2D addresses used to index the pixel memory, and forwarding the received arrays of pixel values to the next module.
D. GPP

The GPP module is used to implement the controller of the AU. It is responsible for managing the quad-tree partitioning of the CTU into CUs and PUs, as well as controlling the execution of the ME of each of the PUs by the accelerators. Furthermore, it is also responsible for communicating with the host GPP and the other PEs, through the usage of the several shared modules described in this chapter, to synchronize the several operations of the ME procedure. For this work, this GPP was implemented by adopting the MB-Lite processor architecture [18].

E. Other Shared Modules

The Pixels DMA controller is responsible for receiving a stream of pixel data from the host GPP and forwarding it to the destination position of the pixel memories. It receives the stream of data in packets, that contain the data from one CTU from the current frame or one 32×32 partition from the reference frame, along with a header which includes the destination address of that packet, the size, and the Translation able configuration data.

The Results Aggregator module offers an individual interface to each PE so that they can write the results without being constrained by the other PEs. These results are sent back to the host main memory through the stream interface.

This Program Memory Configuration module allows the host GPP to configure the internal program memory of each PE. This functionally is essential, so that the several programs can be implemented and tested in the PEs GPPs, without having to reconfigure the hardware.

Finally, the Interrupt Control module allows the PEs to issue interruptions that will be interpreted by the host GPP.

F. Motion Estimation Programming

The ME procedure will be controlled by a program implemented in the MB-Lite GPP, which is responsible for managing the modules inside each PE, by sending commands to the accelerators and interpreting the results. Furthermore, this program is also responsible for managing the CTU partitioning into CTUs and PUs, asking the host GPP for new CTUs and returning the ME results. Naturally, the programming of all these procedures should be easily adaptable to multiple motion search algorithms and strategies. Therefore, a balance had to be achieved between the offered performance and flexibility to implement multiple strategies and algorithms.

Two important structures are used in this program:

- **PUnit** - Used for coordinating the processing steps of one single PU. It contains the PU data, as well as a pointer to a function which implements a step of the ME procedure used to process the PU. By dynamically changing this pointer, multiple steps of the ME can be implemented.

- **PUnitManager** - Used for managing the CTU partitioning scheme and providing the PUs which are ready to be processed by the accelerators.
VI. PROTOTYPING AND EXPERIMENTAL EVALUATION

The complete system was prototyped in a Xilinx Virtex-7 FPGA (XC7VX485T), connected through an 8x PCIe Gen 2 interface to a personal computer equipped with an Intel Core i7 3770K processor, running at 3.5 GHz. In addition, the PCIe framework developed by [17] was used to connect the interfaces of the developed accelerator to the control program running on the Intel Core i7.

The total amount of resources used by the whole temporal prediction subsystem is presented in Table II. As it can be seen, the most used resources are the BRAMs (due to the local pixel memories), followed by the LUTs.

The final architecture was synthesized, mapped, and placed&routed using multiple timing constraints for different clock domains. The adopted PCIe framework [17] has its dedicated clock domain, which requires 250MHz to work properly. For the hardware designed on this work, the place&route step was conducted with a clock constraint of 100MHz. This was required due to the fact that the adopted PCIe framework imposes several constraints regarding the exact placement of its modules. This, combined with the sheer amount of resources used by the 16 PEs of the accelerator, resulted in a more difficult routing optimization process, which did not meet the timing constraints for higher frequencies.

<table>
<thead>
<tr>
<th>Resource Usage of the Whole Temporal Prediction Subsystem.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Whole accelerator (16 PEs) PCIe Framework (16 PEs) Total used (16 PEs)</td>
</tr>
<tr>
<td>Registers 141201 (23,3%) 20886 (3,4%) 168614 (27,8%)</td>
</tr>
<tr>
<td>LUTs 208419 (68,6%) 16859 (5,6%) 238449 (78,5%)</td>
</tr>
<tr>
<td>BRAM36 932 (90,5%) 37 (3,6%) 961 (93,3%)</td>
</tr>
<tr>
<td>Max. Freq. 157 MHz 248 MHz 140/248 MHz(*)</td>
</tr>
<tr>
<td>After Place&amp;Route 100/250 MHz(*)</td>
</tr>
</tbody>
</table>

(*) Different clock domains: PCIe framework requires 250 MHz. The accelerator runs at 100 MHz.

A. Distributed Memory Architecture Analysis

In Fig. 28 it is illustrated the average latency to transfer the pixels of each CTU, and respective search window, from the host GPP to the accelerator, depending on the number of PE. In the background of these plots it is presented a comparison with the amount of time that is available to process each CTU (at 30fps), for the considered number of PEs and resolution. If the resulting latency is below the amount of time available to process the CTU, then the prefetching can be masked by the processing time, while supporting real-time encoding.

As it can be seen, for 128×128 and 192×192 search windows, the distributed memory scheme was able to provide CTUs at a rate compatible with 2160p encoding at 30fps. These are important results, that demonstrate that this distributed memory architecture is compatible with the most demanding requirements of current video coding systems.

B. Accelerators Performance Evaluation

The number of cycles required to process one SAD, by each integer search accelerator, is presented in Table III for the modes 2N×2N (although all the other PU sizes are supported). These values were obtained from simulation.

<table>
<thead>
<tr>
<th>Performance of an Individual Integer Search Accelerator.</th>
</tr>
</thead>
<tbody>
<tr>
<td>PU size</td>
</tr>
<tr>
<td>8×8</td>
</tr>
<tr>
<td>16×16</td>
</tr>
<tr>
<td>32×32</td>
</tr>
<tr>
<td>64×64</td>
</tr>
</tbody>
</table>

The analysis presented in Fig. 29 regards the the performance of the integer search accelerators when all the 16 PEs are working in parallel. In this analysis, the time available to process each CTU is limited by the constraints related with a real-time encoding scenario (30 fps). Therefore, it gives the maximum number of SAD comparisons which can be executed, per PU, in order to meet these requirements (for PUs of mode 2N×2N mode only). Early termination refers to the choice (by the controller) to not partition one CU of size N×N into its sub-CUs of size N/2×N/2, thus saving the time required to analyse those sub-CUs.

As it can be seen in Fig. 29, the considered configuration allows several SADs to be computed for each PU. As expected, it is possible to calculate more SADs for lower resolution...
sizes, because their smaller CTUs number results in more available time to process each CTU and its sub-blocks. The same is observed for higher early termination rates, which correspond to a smaller number of PUs, thus resulting in extra available time to process more SADs for each of the PUs. As illustrated, it is possible to encode 2160p in real-time with the TSS algorithm, provided that the number of PUs to analyse is reduced (for example, by having an average early termination rate of at least 30%).

A similar analysis was conducted for the sub-pixel refinement accelerator. The number of clock cycles required to implement one step of their sub-pixel refinement (half or quarter-precision search), for the PUs of mode $2N \times 2N$, is presented in table IV.

**TABLE IV**

<table>
<thead>
<tr>
<th>PU size</th>
<th>Cycles to complete one refinement step</th>
</tr>
</thead>
<tbody>
<tr>
<td>8×8</td>
<td>74</td>
</tr>
<tr>
<td>16×16</td>
<td>204</td>
</tr>
<tr>
<td>32×32</td>
<td>652</td>
</tr>
<tr>
<td>64×64</td>
<td>2318</td>
</tr>
</tbody>
</table>

By following the same criteria that was considered for the integer search accelerator analysis, the chart presented in Fig. 30 depicts an analysis of the maximum number of sub-pixel refinement operations that can be executed, per PU, in order to support real-time encoding (30 fps) of various formats.

Fig. 30. Maximum number of sub-pixel refinement operations allowed per PU (30fps encoding, by 16 PEs at 100Mhz).

As it can be seen, at least 2 sub-pixel refinement operations are possible for each one of the configurations analysed, which shows that these accelerators are able to execute the sub-pixel refinement step, even when considering 2160p real-time encoding with all $2N \times 2N$ PUs being analysed.

C. Overall Performance Analysis

In this subsection the implementation of two algorithms is analysed: full search block matching and TSS. In each of the two algorithms, only the $2N \times 2N$ PU mode was analysed. This means that, for each CTU, a total of 85 PUs need to be processed. In order to ease the implementation, no dependencies between neighbour PUs were considered for these algorithms.

This full search algorithm was tested with a range of $\pm 16$ pixels in both the X and Y axis. This means that each PU is exhaustively compared to 1024 positions of the reference frame.

Fig. 31. Analysis of the scalability of the full search ME algorithm performance.

Figure 31 shows the performance of this algorithm, in terms of CTUs processed per second, for different amounts of PE working in parallel. As expected, the increase in performance is linear with the number of PEs. This is due to two factors. First, there are no dependencies between neighbour CTUs. Secondly, even with 16 PEs, the time taken to process each CTU significantly longer than any of the prefetching times measured in Fig. 28. This means that, for each new CTU, the PE does not have to wait for the new pixel data, and can start processing it immediately.

The TSS algorithm requires each PU to be compared to 8 positions of the search window, in each of the three steps. This, added to the location at the origin of the referential, gives a total of 25 SADs for each PU.

Fig. 32. Analysis of the scalability of the three step search ME algorithm performance.

Figure 31 shows the attained performance of this algorithm. In this case, the increase in performance also varies linearly with the number of PEs, due to the same reasons of the full search algorithm. With 16 PEs this architecture is able to execute the TSS ME algorithm to encode the 720p resolution in real time. The reason for not being able to process more CTUs is because of the controller implemented in the MB-Lite GPP. Higher resolutions might be achieved by increasing the number of PEs, increasing the frequency, reducing the number of analysed PUs, or implementing a less demanding ME algorithm. Alternatively, it could be considered the implementation on an ASIC, instead of the adopted FPGA device, which can achieve speedups as high as $4 \times$, according
to [19]. This, together with a better GPP, would perpertivcitate the encoding of the 2160p resolution in real-time.

VII. Conclusions and Future Work

In this thesis, a distributed prediction scheme for the inter-frame prediction step of the HEVC standard was presented, implemented and evaluated. Several aspects were considered, such as the support of the encoding of high definition formats in real-time, the compatibility with multiple ME algorithms and strategies, the scalability of the architecture and the support for dedicated accelerators.

The proposed scheme involved the usage of a scalable number of AUs working in parallel. These conceptual AUs were implemented with a module denoted as PE, able to execute of the ME procedure of different CTUs. These PE contain highly optimized accelerators, which were demonstrated to be compatible with real-time encoding of 2160p at 30fps, even when considering that the prototyped implementation is limited to an operating frequency of 100MHz. Furthermore, the ME processing steps inside each PE are controlled by a local GPP. A program for processing the ME was proposed, which can be easily adapted to implement different ME algorithms and strategies.

In addition, to efficiently provide the pixel data to the accelerators, an optimized distributed memory mechanism was also proposed. This mechanism employs data prefetching and data reuse techniques, which allows the architecture to greatly reduce the impact which the data transfers have in the overall performance. The conducted evaluation showed that this distributed pixel memory mechanism is able to transfer the pixel data, from the main encoder (at the host computer) to the local pixel memories, at a rate compatible with the encoding of the 2160p resolution (4K-UHD) at 30fps.

Finally, in order to demonstrate the compatibility of the proposed architecture with different algorithms and search strategies, two ME algorithms were implemented and evaluated: full search and TSS.

A. Future Work

In order to improve its overall performance, some further work can be considered.

First, the computational performance of the GPP that is embedded in each PE can be improved, in order to be able to match with the high performance of the accelerators and of the distributed pixel memory scheme. Secondly, the whole architecture may be prototyped in an ASIC technology to provide a significant increase of the operating clock frequency, which will provide the PEs and the accelerators with more time to process the CTUs. Thirdly, more complex ME algorithms may be implemented. In particular, some of the recently proposed ME algorithms, such as the EPZS [5] or TZZS [6], which provide a very good compression quality. Finally, the proposed architecture for interframe prediction shall be incorporated with a complete HEVC encoder, in order to provide a final product which effectively accelerates its encoding procedure.

REFERENCES


