Flexible and Scalable Implementation of H.264/AVC Encoder for Multiple Resolutions Using ASIPs

Hong Chinh Doan Haris Javaid Sri Parameswaran
School of Computer Science and Engineering
University of New South Wales, Sydney, Australia
Email: {hcdo420, harisj, sridevan}@cse.unsw.edu.au

***Abstract*—Real-time encoding of video streams is computationally intensive and rarely carried out at high resolutions. In this paper, for the first time, we propose a platform for H.264 encoder which is both flexible (allows software upgrades) and scalable (supports multiple resolutions), and supports high video quality (by using both intraprediction and interprediction) and high throughput (by exploiting slice-level and pixel-level parallelisms). Our platform uses multiple Application Specific Instruction set Processors (ASIPs) with local and shared memories, and hardware accelerators (in the form of custom instructions). Our platform can be configured to use a particular number of ASIPs (slices per video frame) for a specific video resolution at design-time. The MPSoC architecture is automatically generated by our platform and the H.264 software does not need any modification, which enables quick design space exploration. We implemented the proposed platform in a commercial design environment, and illustrated its utility by creating systems with up to 170 ASIPs supporting resolutions up to HD1080. We further show how power gating can be used in our platform to save energy consumption.**

I. INTRODUCTION

Recent years have seen a radical increase in the use of multimedia devices, especially video related devices such as high-end interactive displays, gaming consoles, video conferencing, High Definition (HD) TVs, etc. A major component of such video devices is a video encoding and decoding (codec) engine. H.264/Advanced Video Coding (AVC) [1] is one of the most commonly used codecs because it provides good video quality at substantially lower bit rates compared to previous standards such as H.263, MPEG-2 or MPEG-4, as well as flexibility in trading-off computational complexity with video quality [2]. Users of video devices expect better video quality at higher resolutions (such as HD720 and HD1080) and support for multiple resolutions in just a single device for ease of use and minimal cost. Additionally, they expect to have quick software fixes and upgrades rather than buying a new device every few months. Therefore, a flexible (support for software upgrades), scalable (support for multiple resolutions) H.264 video codec that can deliver high video quality and throughput in real-time even at higher resolutions needs to be designed with short time-to-market, which is a challenging task.

Motivational Example. An H.264 encoder consists of several sub-kernels such as Intra Prediction (IP); Motion Estimation(ME) and Motion Compensation (MC); (Inverse) Discrete Cosine Transform ((I)DCT); Quantization (QUANT); DeQuantization (DQUANT); Deblocking Filter (DF); and, Entropy Coding (EC). The frames of the incoming video are processed by these sub-kernels one after the other. For each MacroBlock (MB) of the current frame, the H.264 encoder starts with either MB prediction from current frame if intraprediction is to be used or motion estimation from reference frame(s) if interprediction is to be used [1]. Motion estimation is considered the most computationally intensive sub-kernel [3] because it searches for the best possible matching MB in reference frame(s) using Sum of Absolute Differences (SAD). Motion estimation can either be performed using the pixels of the reference MB (integer-pel) or using interpolated pixels (such as half, quarter, etc.) of the reference MB (fractional-pel) [1]. Previous research has shown that fractional-pel motion estimation results in better video quality and lower bit rates (up to 50% [4]). However, such an improvement comes at the cost of extra computational complexity.

Figure 1 reports the time taken to process a frame when intraprediction, integer-pel motion estimation and fractional-pel motion estimation are performed on the same frame. Additionally, Figure 1 reports the processing time of a frame for four different resolutions: CIF, 4CIF, HD720 and HD1080. In these experiments, same search window is used

978-3-9815370-2-4/DATE14/©2014 EDAA

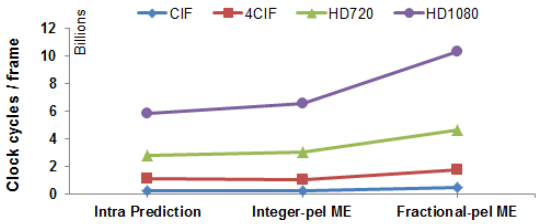

Fig. 1. Execution time of different prediction modes for different resolutions in H.264 kernel.

across intraprediction and motion estimation, and across all resolutions. Additionally, the H.264 encoder is implemented on a 16 processor MultiProcessor System-on-Chip (MPSoC; details can be found in Section III-B). It is evident that use of fractional-pel motion estimation increases the computational complexity manifold (up to 1.8 times). Furthermore, the increase in computational complexity is more severe at higher resolutions. This example shows that a naive implementation of H.264 encoder will not suffice the goals of high video quality and throughput in real-time as well as flexibility and scalability.

Previous research has used several adhoc methodologies to cope with (a subset of) the challenges of a flexible, scalable, high video quality and throughput H.264 encoder. For example, the authors of [5] implemented an H.264 encoder for QCIF resolution only, while the authors of [2] implemented H.264 encoder for HD1080 resolution at 30 fps without the support for intraprediction and interprediction. [6] implemented H.264 encoder on a 64 processor MPSoC for HD1080 resolution at 30 fps with intraprediction only. The work in [7] also implemented an H.264 encoder with intraprediction only, but for multiple resolutions from QCIF to HD1080. Iwata et al. [8] implemented an H.264 encoder with fractional-pel motion estimation for HD1080 resolution at 30 fps. Both [7], [8] proposed Application Specific Integrated Circuits (ASICs), which are inflexible and require high redesign effort. The above works either lack one or more of the aforementioned goals of flexibility (support for software upgrades) and scalability (support for multiple resolutions) without little or no redesign effort, and high video quality (fractional-pel motion estimation) and throughput in real-time.

To address the above challenges, in this paper, we propose a hardware-software codesign platform where slice-level and pixel-level parallelisms at the application-level are uniquely combined with multiple Application Specific Instruction set Processors (ASIPs) and hardware accelerators (in the form of custom instructions) at the architecture-level. **Our key contributions are:**

- We combine slice-level and pixel-level parallelisms in a unique manner to seamlessly scale across multiple resolutions.
- At architecture-level, we use an MPSoC consisting of multiple ASIPs with hardware accelerators (in the form of custom instructions). The ASIPs process slices in parallel using their hardware accelerators, simultaneously benefiting from both slice-level and pixel-level parallelisms for high throughput. The flexibility comes from the use of ASIPs which are programmable.
- To maximally reduce redesign effort, our platform automatically generates the MPSoC architecture while the H.264 kernel code does not need any modification when the number of ASIPs (slices) and video resolution is changed. This enables quick design space exploration for optimization.

II. RELATED WORK

There is a large body of work on implementation of complete or a part of H.264 encoder. The following paragraphs report the most relevant complete implementations, focusing on slice-level parallelism (for high throughput and scalability), pixel-level parallelism (for high throughput), intraprediction and interprediction (for video quality), and ASIC or MPSoC implementation (for flexibility and scalability).

Iwata et al. [8] proposed an ASIC with a dual MB pipeline to process two rows of a frame at the same time, exploiting a form of slice-level parallelism. They also implemented fractional-pel motion estimation for better video quality and targeted HD1080 resolution at 30 fps, without the support for multiple resolutions. Li et al. [7] proposed an ASIC for HD1080 resolution, 30 fps and multiple resolutions but without interprediction, and thus compromising on video quality. The authors of [9] proposed an ASIC to support multiple resolutions with both intraprediction and interprediction, but without any form of slice-level parallelism. In general, ASICs guarantee real-time throughput with low power consumption; however, they are inflexible, and require high (re)design efforts.

Unlike ASICs, MPSoCs provide a flexible (programable) implementation platform and have been used in [2], [6], [10], [11] for implementation of H.264 encoder. The authors of [12] did not use any prediction, while the authors of [6] used only intraprediction, and thus compromising on video quality. The authors of [10], [11] exploited task-level parallelism (where sub-kernels of H.264 encoder are executed on separate processors) rather than slice-level parallelism as in our work. Since slices are processed independently of each other, slice-level parallelism inherently enables scalability in the MPSoC as the number of processors can be changed without major redesign effort (see Section III-B).

ASIPs have recently emerged as a viable implementation platform for video applications due to their programmable nature and capability of adding hardware accelerators (in the form of custom instructions) [13]. The works in [14], [5], [15] used a single ASIP, while the work in [16] used multiple ASIPs to implement just the motion estimation sub-kernel rather than the complete H.264 encoder. To the best of our knowledge, we are the first to propose an MPSoC architecture with multiple ASIPs and hardware accelerators, augmented with slice-level and pixel-level parallelisms as a flexible, scalable implementation platform for H.264 with support for multiple resolutions.

III. HARDWARE-SOFTWARE CODESIGN PLATFORM FOR H.264 ENCODER

There are several factors that need to be considered for flexible and scalable implementation of an H.264 encoder with minimal redesign effort. For example, one should use the kind of parallelism that will require little or no software modifications when resolution increases from CIF to HD1080. Likewise, if the number of processors are changed to support higher resolutions or better motion estimation (fractional-pel), then the changes in the architecture should be minimal. We propose a hardware-software codesign platform where slice-level and pixel-level parallelisms at the application-level are combined with multiple ASIPs and hardware accelerators (in the form of custom instructions) at the architecture-level.

A. H.264 Encoder

Figure 2(a) shows the typical structure of an H.264 encoder, where the incoming stream is encoded by its sub-kernels. The first sub-kernel is either IP or ME and MC depending upon whether intraprediction or interprediction is used for the current frame. Afterwards, the residual data is computed and forwarded to DCT and QUANT sub-kernels for transformation and quantization, followed by entropy coding. From DCT sub-kernel, another path is used to reconstruct the reference frames. The reconstruction path consists of inverse quantization and transformation which are performed at the DQUANT and IDCT sub-kernels respectively. Finally, the DF sub-kernel smooths the MBs of the reference frame. The traditional parallelization approaches execute the sub-kernels of the H.264 encoder on different processors to exploit task- and pipeline-level parallelism [10], [11]. Unlike those approaches, as shown in Figure 2(b), we arrange the H.264 encoder in three stages where the H.264 encoder is considered a single kernel (rather than multiple sub-kernels) in the second stage. The first stage is responsible for providing input video stream to the second stage, while the third stage reads the output of the second stage to produce the final bit stream.

Our proposed arrangement of H.264 encoder benefits from the following: (1) The MBs of the current frame can be processed in parallel inside the H.264 kernel except for the EC sub-kernel. Therefore, the current frame can be partitioned into several slices where each slice contains a consecutive number of MBs in raster scan order which is beneficial for MB prediction [1]. These slices can then be processed in parallel by multiple instances of the H.264 kernel, exploiting slice-level parallelism. For example, the IN stage in Figure 2(b) reads a frame, partitions it into N slices and allocates those slices to N H.264 kernels (executed on N separate ASIPs, details in Section III-B) in the second stage. Each H.264 encoder processes the MBs in its allocated slice in a sequential manner and outputs the (partial) bit stream of the encoded slice. Finally, the OUT stage reads the (partial) bit streams of all the N slices and combines them to generate the final bit stream for the current frame. The scalability of our approach comes from the fact that the whole frame or each MB itself could be considered a slice, with no theoretical limits on the granularity of the slice. Additionally, the number of slices and hence the number of times H.264 kernel is to be duplicated (N) can be selected by the designer depending on his/her resolution and throughput goal such as HD1080 at 30 fps. (2) The redesign and verification efforts are very high for an H.264 kernel whose sub-kernels are executed on separate processor, and multiple resolutions, execution modes and algorithmic updates have to be supported. Since we treat H.264 kernel as a single kernel and duplicate it, it can be independently modified and tested which will reduce the redesign and verification efforts leading to better flexibility and scalability of our approach.

B. MPSoC Architecture

To achieve high throughput in real-time along with flexibility and scalability, we use an MPSoC architecture with ASIPs, local caches and memories, and shared memory as shown in Figure 3. The MPSoC has one ASIP each for IN and OUT stages, and N ASIPs for N instances of H.264 kernel. Each ASIP is equipped with hardware accelerators in the form of custom instructions (explained in Section III-D), and has private L1 instruction and data caches (IC and DC), connected to a local memory. In addition, all the ASIPs are connected to a shared memory which bypasses their local memory hierarchy. The instructions and local data are kept in the local memory of an ASIP, while the shared data is kept in shared memory. The shared memory is partitioned into three predefined regions: (1) for current frame, (2) for reference, reconstructed and sub-pixel (half, quarter, etc.) interpolated frames, marked as "Reference Frames", and (3) for (partial) bit streams of the slices of current frame marked as "STREAM". The IN stage writes the current frame to shared memory, while the OUT stages reads (partial) bit streams of the slices of the current frame from the shared memory.

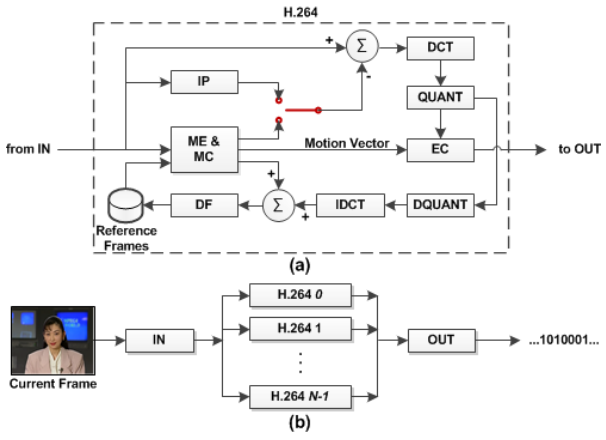

Fig. 2. Our proposed arrangement of H.264 encoder.

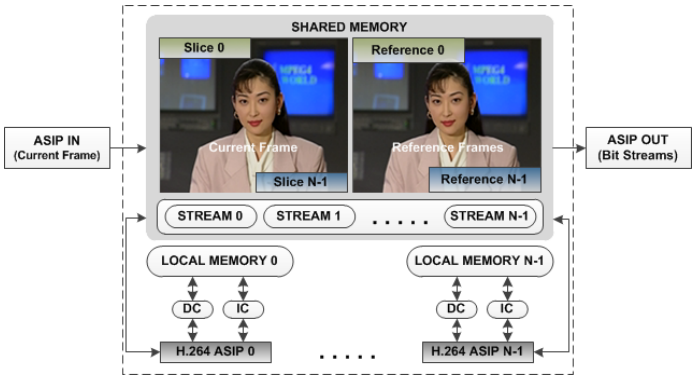

Fig. 3. Our MPSoC architecture for H.264 encoder.

During processing of one frame, each H.264 ASIP reads an MB from the slice of the current frame allocated to it into its local memory. If intraprediction is used, then the MB is processed locally by storing residual MB and quantized MB in its local memory to avoid relatively higher latency of shared memory. Additionally, the local memory hierarchy benefits from hits in private L1 data cache. If interprediction is used, then the reference MBs within the search window are read from shared memory without storing them locally. This is because the amount of data in reference MBs grows significantly when larger search windows and fractional-pel motion estimation is used. Additionally, search windows of different MBs overlap [16] and storage of reference MBs in local memories means that the overlapping parts are redundantly stored at multiple locations, increasing the size of local memories. Therefore, during interprediction, reference MBs are directly read from shared memory, while residual MB and quantized MB are stored locally. In both intraprediction and interprediction, the reconstructed MB is directly stored in shared memory for deblocking filter to produce and store sub-pixel interpolated frame in shared memory. Each ASIP also encodes all the residual MBs of its slice, creates a packet of the (partial) bit stream of the slice and writes it to the shared memory. Note that this distribution of data among local and shared memories does not affect scalability of our approach as the ASIPs can automatically calculate the addresses of the shared memory regions that belong to them (explained in Section III-C).

Algorithm 1: Pseudo code of our H.264 kernel

Input: configuration file
ip: index of the H.264 ASIP

```
1  Read configuration file;
   Allocate slice based upon ip and resolution;
   Compute encoding parameters;
2  for all frames do
3      Wait for signal from IN stage;
4      for all MBs in the slice do
5          Initialize MB;
6          if Intra MB then
7              Perform Luma (Y) intraprediction;
               Compute residual MB of Y;
               Perform DCT, QUANT, DQUANT and IDCT;
               Compute reconstructed MB of Y;
8              Perform Chroma (U, V) intraprediction;
               Compute residual MB of U and V;
               Perform DCT, QUANT, DQUANT and IDCT;
               Compute reconstructed MB of U and V;
9          else
10             Perform Y integer-pel ME and fractional-pel ME;
               Compute residual MB of Y using motion vector;
               Perform DCT, QUANT, DQUANT and IDCT;
               Compute reconstructed MB of Y;
11             Compute residual MB of U and V using motion vector;
               Perform DCT, QUANT, DQUANT and IDCT;
               Compute reconstructed MB of U and V;
12         Perform entropy encoding of residual MB;
13     Copy encoded slice from local memory to shared memory;
14     Signal OUT stage to read (partial) bit stream of slice AND
       wait for signal from OUT stage;
       Signal IN stage;
15     for all MBs in the slice do
16         Perform Y sub-pixel interpolation on reconstructed MB;
```

C. Software Details

Algorithm 1 demonstrates the pseudo code of our H.264 kernel which is executed on each of the H.264 ASIPs. While conforming to the H.264 standard, we modified and optimized the structure of the kernel for seamless scalability across multiple resolutions. The input arguments include a configuration file and index of the H.264 ASIP. The configuration file provides the video resolution and the total number of H.264 ASIPs, in addition to H.264 encoding parameters. An example of the configuration file for HD1080 resolution and 36 H.264 ASIPs is shown in Table I. This information is used in line 1 to automatically allocate the slice of the current frame and (partial) bit stream region in shared memory to the ASIP on which the H.264 kernel is executing (explained later in Algorithm 2), and to compute the encoding parameters.

The main loop (lines 2 – 16) of the H.264 kernel processes one frame at a time. At the start, the H.264 kernel waits for the ready signal (line 3) which is set after writing of the current frame by the IN stage. Once the current frame is available, the H.264 kernel processes the MBs in its slice one by one until entropy coding is needed (lines 4 – 12). Once the whole slice has been encoded, it is copied to the corresponding location in the "STREAM" region of shared memory (line 13). This marks the end of the slice processing, and thus the H.264 ASIP signals OUT stage about the availability of its (partial) bit stream and waits for a response (line 14). The OUT stage waits for all the H.264 ASIPs to finish processing of their slices, and then signals those ASIPs to continue their execution. This two way signaling mechanism acts as a synchronization barrier between the H.264 ASIPs to ensure that all the slices of the current frame are processed before the ASIPs continue further. Therefore, after receiving response from the OUT stage, the H.264 ASIP signals IN stage to write the next frame from the input video. While IN stage writes the new frame, all H.264 ASIPs perform sub-pixel (half, quarter, etc.) interpolation on the reconstructed frame for fractional-pel motion estimation. This marks the end of the processing of the current frame, and the H.264 kernel is repeated. The signaling mechanism between the ASIPs is implemented using status flags in shared memory.

In order to correctly read the slice from and write the (partial) bit stream to the shared memory, an H.264 ASIP needs to determine the number of MBs in its slice ($sliceSize$) and the offset of its slice ($sliceStart$ in terms of number of MBs) in the current frame. This depends on the index of the H.264 ASIP, the total number of H.264 ASIPs and the video resolution. Algorithm 2 computes such information automatically. At first (lines 1 – 2), the average number of MBs per slice and the number of trailing MBs are calculated. If there are no trailing MBs, then the slices are of equal size (lines 3 – 5). Otherwise, each H.264 ASIP is allocated a trailing MB starting from ASIP 0 until the trailing MBs are exhausted (lines 7 – 12). Since the number of trailing MBs cannot exceed the number of H.264 ASIPs (due to the remainder operation at line 2), such a distribution ensures a maximum difference of 1 MB across all the slices, keeping the workload of H.264 ASIPs relatively balanced in the number of MBs. Once each ASIP calculates its $sliceSize$ and $sliceStart$, it can access the corresponding regions of the shared memory using the start address of the current frame (which is

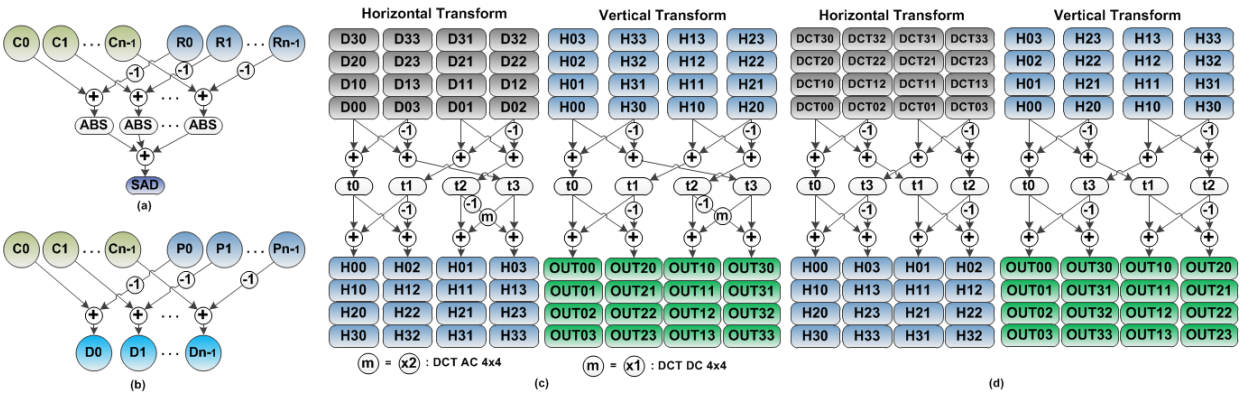

Fig. 4. Hardware accelerators (custom instructions) used in our experiments.

Algorithm 2: Slice Allocation

Input: N: total number of H.264 ASIPs
N_{MB}: total number of MBs in a frame
$streamSize$: the maximum size of slice bit stream
ip: index of the H.264 ASIP

1 $q = \lfloor \frac{N_{MB}}{N} \rfloor$;
2 $r = N_{MB}\%N$;
3 **if** $r == 0$ **then**
4 $sliceSize = q$;
5 $sliceStart = ip \times q$;
6 **else**
7 **if** $ip < r$ **then**
8 $sliceSize = q + 1$;
9 $sliceStart = ip \times q$;
10 **else**
11 $sliceSize = q$;
12 $sliceStart = (q + 1) \times r + (ip - r) \times q$;
13 $streamStart = ip \times streamSize$

known a priori). The offset of slice bit stream in the "STREAM" region of the shared memory is calculated in line 13 where the maximum size of slice bit stream ($streamSize$) is provided in the configuration file. For reference frame(s), reconstructed frame and sub-pixel interpolated frame, a method similar to allocation of slice from current frame is adopted.

Frame Width (W)	1,920
Frame Height (H)	1,080
Number of IN ASIPs	1
Number of H.264 ASIPs	36
Number of OUT ASIPs	1

TABLE I
AN EXAMPLE OF A CONFIGURATION FILE.

The allocation of slices and bit stream regions is simple and easy to implement. More importantly, it is scalable because the algorithm is parameterized to use values of parameters from the configuration file. Thus, our H.264 encoder is scalable across different video resolutions, MB size, search window, number of H.264 ASIPs, etc. For example, scaling an implementation with 32 H.264 ASIPs to 64 H.264 ASIPs does not require any modifications to software – only the configuration file needs to be updated. Moreover, we automatically generate the MP-SoC architecture described in Section III-B from the configuration file, which further minimizes the redesign effort and makes our approach more scalable.

D. Hardware Accelerators

In addition to slice-level parallelism, a huge amount of pixel-level parallelism is available in an H.264 kernel. Many fundamental operations such as Sum of Absolute Differences (SAD), Discrete Cosine Transform (DCT), etc. allow parallel processing of all (or a subset) of the pixels in an MB. Since these operations are frequently executed, their accelerated implementations can significantly improve throughput. Figure 5 reports the distribution of time spent in different sub-kernels of an H.264 kernel when an MB is either intrapredicted or interpredicted. It is evident that EC, IP and DCT are the most computationally intensive sub-kernels for an intrapredicted MB, while ME, EC, IDCT and DCT are the most intensive sub-kernels for interpredicted MB. Based on this observation, we opted to implement hardware accelerators (in the form of custom instructions for H.264 ASIPs) for SAD operation (used in both IP and ME), residual MB operation (used in DCT), DCT operation, and IDCT operation. We did not consider hardware accelerator for EC because the encoding process has a lot of data dependencies with little pixel-level parallelism. Note that this does not limit the scalability of our approach as a designer can implement any hardware accelerator or choose from a set of different hardware accelerators based upon the available hardware budget. The following is an example of hardware accelerators used in our experiments.

SAD accelerator. The accelerator for SAD operation is shown in Figures 4(a) where the inputs are current MB and reference MB. The reference pixels, R_0 to R_{n-1}, are subtracted from the corresponding current pixels, C_0 to C_{n-1}. After that, an absolute operation is applied on all the differences, followed by their addition to compute SAD. The number of pixels to be processed in parallel, n, can be either 8 or 64 (for U and V) and 16 or 256 (for Y). We used $n = 8$ for U and V, and $n = 16$ for Y, which corresponds to processing of one MB row in the accelerator.

Residual data accelerator. Figure 4(b) shows the hardware accelerator to compute the residual MB. In this operation, each pixel of the predicted MB, P_0 to P_{n-1}, is subtracted from the corresponding pixel of the current MB, C_0 to C_{n-1}, to compute the residual values, D_0 to D_{n-1}. Like SAD accelerator, we implemented $n = 8$ for U and V, $n = 16$ for Y.

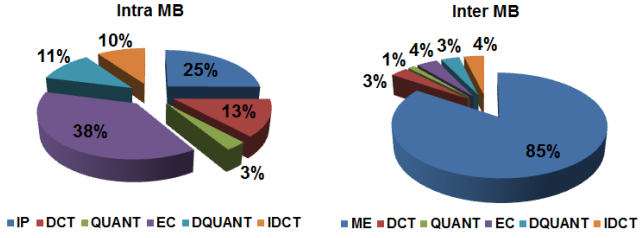

Fig. 5. Distribution of workload among H.264 sub-kernels.

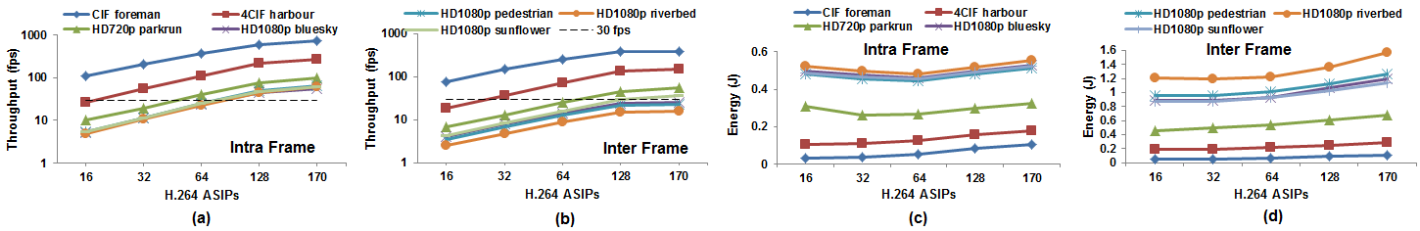

Fig. 6. Throughput and energy consumption vs. number of ASIPs for different resolutions.

DCT accelerator. The fundamental DCT operation is performed on a 4×4 block where DCT is first computed horizontally (on rows) and then vertically (on columns). Figure 4(c) illustrates typical butterfly structure of a DCT where the input is marked $D00$ to $D33$ while the output is marked $OUT00$ to $OUT33$. $t0$ to $t3$ are intermediate values and $H00$ to $H33$ is the result of the horizontal transform. The parameter m equals 2 and 1 when DCT on Alternative Current (AC) and Direct Current (DC) is computed, respectively.

IDCT accelerator. Figure 4(d) illustrates the operation of IDCT on 4×4 block for DC coefficients. The input data is similar to the DCT operation; however, the butterfly structure is different.

IV. EXPERIMENTS AND RESULTS

Experimental Setup. We implemented the proposed hardware-software codesign platform for H.264 encoder in a commercial design environment from Tensilica [17]. We used Xtensa LX4.0 family of processors along with the RD-2012.5 tool suite which contains XTensa Modeling Protocol (XTMP), Instruction Set Simulator (ISS), Xtensa energy estimator (Xenergy), Xtensa C and C++ compiler (XCC) and Tensilica Instruction Extension (TIE) compiler. XTMP is an API and run-time environment for rapid creation of multiprocessor systems with local and shared memories. XTMP uses ISS and Xenergy to cycle-accurately simulate a multiprocessor system to provide its performance and energy consumption. XCC is used to compile C and C++ programs for Xtensa processors, which can also recognize custom instructions (for hardware accelerators) written in TIE language. The custom instructions are compiled through TIE compiler and then XCC automatically uses those instructions without the need for modifying the C/C++ code.

The ASIP used in our implementation was configured for a 5-stage pipeline, 128-bit Processor Interface (PIF), 128-KB 4-way set associative instruction and data caches, and 16 MegaBytes of local memory. The area of the ASIP including the hardware accelerators (described in Section III-D) is 205K gates. Additionally, 96 MegaBytes of shared memory was used in the MPSoC. Our MPSoC was executed at a frequency of 1 GHz and configured for a 45nm technology for computation of energy consumption. We used the following video sequences and resolutions: CIF foreman, 4CIF harbour, HD720p parkrun, and HD1080p bluesky, pedestrian, riverbed and sunflower for rigorous evaluation of our proposed platform.

We used the H.264 reference software [18] as the H.264 kernel in our platform. The H.264 reference software was modified to exploit slice-level parallelism and pixel-level parallelism (through the use of custom instructions). UMHexagon [3] is used as the motion estimation algorithm.

Design Space Exploration. It took 6 man-months to design and implement the proposed hardware-software codesign platform. Afterwards, the configuration file is set with different parameter values to realize different systems within a few minutes without the need for a major redesign effort. Therefore, our platform inherently allows quick design space exploration for its optimization. Here, we illustrate design exploration for two modes of H.264 (intraprediction and iterprediction of frames), number of H.264 ASIPs and four different resolutions (CIF, 4CIF, HD720 and HD1080). Note that a designer can also explore other parameters such as motion estimation algorithms, cache configurations, hardware accelerators, etc. with our platform.

Figure 6(a)(b) plots the throughput on y-axis (in fps) against the number of H.264 ASIPs (from 16 to 170) for different video sequences and resolutions. The left and right graphs report the throughput for intraprediction and interprediction respectively. The throughput generally increases with an increase in the number of H.264 ASIPs. More importantly, a designer can select the number of H.264 ASIPs based upon his/her requirement. For example, if an H.264 encoder supporting 30 fps up to HD720 resolution with only intraprediction is required, then 70 ASIP MPSOC should be deployed in the MPSoC. Likewise, if an H.264 encoder supporting 30 fps up to HD1080 resolution with both intraprediction and interprediction is required, then 170 ASIPs should be used. A similar procedure can be adopted for energy consumption which is reported in Figure 6(c)(d) where the energy consumption is reported on y-axis in Joules and the number of H.264 ASIPs on x-axis. For the aforementioned parameters, we also explored throughput against energy consumption and area footprint, but omitted those results due to limited space.

The scalability of our platform comes from the exploitation of slice-level parallelism which scales well with an increase in the resolution. However, encoding slices of a frame independently results in higher bit rates compared to encoding of the whole frame (that is, one slice) [1]. Therefore, we performed exploration on the size of final bit stream against the number of slices per frame. Figure 7 plots the bit stream size on the y-axis normalized to the bit stream size from the 16 H.264 ASIP MPSoC against the number of slices per frame on the x-axis. The size of the bit stream increases by a maximum of 23% from 16 to 170 slices per frame, which is the cost of scalability of our platform. A designer can select the MPSoC which conforms to his/her budget of increase in bit rate.

Finally, we present the power consumption of our platform in Figure 8. As expected, the power consumption increases with an increase in the number of H.264 ASIPs, up to 24 Watts for 170 ASIPs. Note that we tradeoff power consumption with flexibility and scalability unlike ASICs where flexibility is compromised for low power consumption. Furthermore, high quality and throughput (fractional-pel

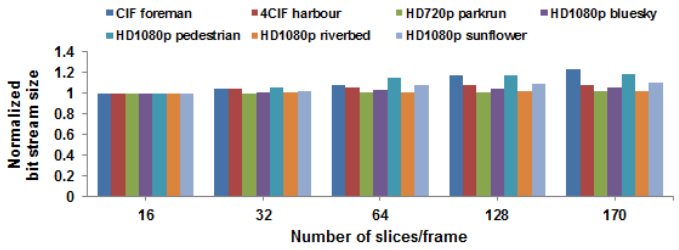

Fig. 7. Size of bit stream vs. number of ASIPs for different resolutions.

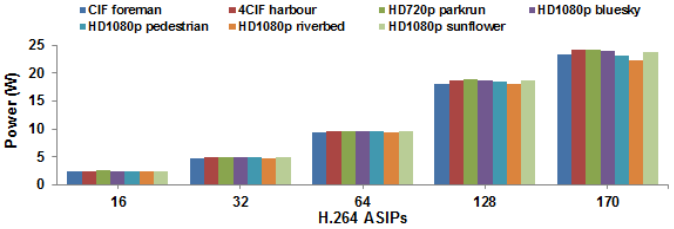

Fig. 8. Power consumption vs. number of ASIPs for different resolutions.

motion estimation, HD1080, 30 fps) is required in high-end devices such as JVC or SONY professional media camcorders which are fueled by power supplies rather than batteries. Also, other non-ASIC implementations of H.264 encoder for high-end devices [19] report similar power consumption.

It is evident from the above results that our proposed hardware-software codesign platform serves as a flexible, scalable implementation platform for H.264 encoder where high video quality and throughput in real-time is possible. Additionally, it allows quick design space exploration for several optimizations.

Energy Optimization. In this subsection, we illustrate how our platform can be extended to use power-gating for energy minimization. An H.264 ASIP waits for other ASIPs to finish their corresponding slices before continuing to the next frame, as illustrated in line 14 of Algorithm 1. This idleness of the ASIP can be exploited to reduce energy consumption. We implement a simple power gating mechanism where the OUT stage deactivates an H.264 ASIP once it finishes writing of its (partial) bit stream, and reactivates it when all the ASIPs have processed the slices of the current frame. Figure 9 reports the reduction in energy consumption of the MPSoC, where an energy overhead of 250 nJ for power gating is used [20]. For a given resolution, the energy saving improves with an increase in ASIPs because more ASIPs will remain idle in the MPSoC. However, the improvement is small especially for higher resolutions which means that the idle periods are short and the ASIPs are almost fully utilized. For example, the energy saving improves from 1.3% to 3.8% only for riverbed HD1080 video sequence when ASIPs increase from 16 to 170, signifying the fact that the idle periods are very short. This case study shows that our platform is flexible enough to allow further research and development.

Discussion. The experiments above illustrate that our platform is flexible and scalable, and can serve as a development and exploration platform. Theoretically, it can be extended to even higher resolutions such as Ultra High Definition (UHD), and Scalable Video Coding (SVC). However, in our experiments, we observed a limit on the scalability of our platform. In Figure 6(a)(b), the improvement in throughput from 128 to 170 H.264 ASIPs reduces especially when interprediction is used. Figure 10 reports the improvement in throughput when the MPSoC's shared memory latency is changed from 25 clock cycles to 1 clock cycle, essentially treating the shared memory as a shared cache. In general, the improvement in throughput reduces with an increase in the number of ASIPs. This means that if the number of ASIPs is small, then the bottleneck is in actual read/write operation of the shared memory. However, as the number of ASIPs increases, the bottleneck shifts to the contention in gaining access to the shared memory rather than the actual read/write operation. From this observation, we conclude that multi-banked cache and shared memory should be used with clever arrangement of data to support further scalability of our platform. We aim to explore this in future.

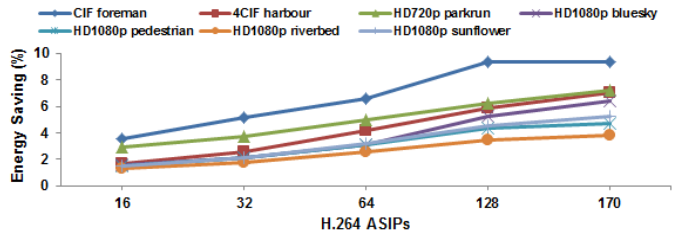

Fig. 9. Energy savings from application of power-gating.

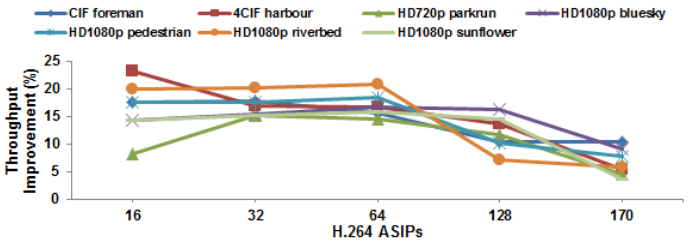

Fig. 10. Throughput improvement when shared memory access latency is reduced from 25 clock cycles to 1 clock cycle.

V. CONCLUSION

In this paper, we proposed a platform for H.264 encoder which exploits slice-level parallelism with multiple ASIPs and pixel-level parallelism with hardware accelerators. We uniquely combine these parallelisms with multiple ASIPs and hardware accelerators to offer a platform that is flexible (allows software upgrades) as well as scalable (supports multiple resolutions). Our platform generates the MPSoC architecture without manual intervention and the H.264 kernel software does not need any modification when the number of ASIPs and/or video resolution changes. Finally, we illustrated that our platform could contain up to 170 ASIPs for real-time encoding of an HD1080 video stream at 30 fps (with intraprediction, integer-pel and fractional-pel motion estimation).

REFERENCES

[1] I. E. G. Richardson, "H.264 and mpeg-4 video compression," 2003.

[2] Z. Xiao, S. Le, and B. Baas, "A fine-grained parallel implementation of a h.264/avc encoder on a 167-processor computational platform," in *Signals, Systems and Computers (ASILOMAR)*, pp. 2067–2071, 2011.

[3] Z. Chen, J. Xu, Y. He, and J. Zheng, "Fast integer-pel and fractional-pel motion estimation for h.264/avc," *Journal of Visual Communication and Image Representation*, vol. 17, no. 2, pp. 264 – 290, 2006.

[4] T. Wiegand, G. Sullivan, G. Bjntegaard, and A. Luthra, "Overview of the h.264/avc video coding standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 560–576, 2003. cited By (since 1996)3384.

[5] T. Dias, N. Roma, and L. Sousa, "H.264/avc framework for multi-core embedded video encoders," in *System on Chip (SoC), 2010 International Symposium on*, pp. 89–92, 2010.

[6] I. A. Jose Parera-Bermudez, Javier Casajus-Quiros, "Video encoder implementation on tilera's tilepro64 multicore processor," 2013.

[7] G.-L. Li, T.-Y. Chen, M.-W. Shen, M.-H. Wen, and T.-S. Chang, "135-mhz 258-k gates vlsi design for all-intra h.264/avc scalable video encoder," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 21, no. 4, pp. 636–647, 2013.

[8] K. Iwata, S. Mochizuki, M. Kimura, T. Shibayama, F. Izuhara, H. Ueda, K. Hosogi, H. Nakata, M. Ehama, T. Kengaku, T. Nakazawa, and H. Watanabe, "A 256 mw 40 mbps full-hd h.264 high-profile codec featuring a dual-macroblock pipeline architecture in 65 nm cmos," *Solid-State Circuits, IEEE Journal of*, vol. 44, pp. 1184 –1191, april 2009.

[9] Y.-K. Lin, D.-W. Li, C.-C. Lin, T.-Y. Kuo, S.-J. Wu, W.-C. Tai, W.-C. Chang, and T.-S. Chang, "A 242mw, 10mm21080p h.264/avc high profile encoder chip," in *Proceedings of the 45th annual Design Automation Conference*, DAC '08, (New York, NY, USA), pp. 78–83, ACM, 2008.

[10] M. Bariani, P. Lambruschini, and M. Raggio, "An efficient multi-core simd implementation for h.264/avc encoder," *VLSI Des.*, vol. 2012, pp. 3:3–3:3, Jan. 2012.

[11] S. Chen, S. Chen, H. Gu, H. Chen, Y. Yin, X. Chen, S. Sun, S. Liu, and Y. Wang, "Mapping of h.264/avc encoder on a hierarchical chip multicore dsp platform," in *High Performance Computing and Communications (HPCC), 2010 12th IEEE International Conference on*, pp. 465–470, 2010.

[12] Z. Xiao and B. Baas, "A 1080p h.264/avc baseline residual encoder for a fine-grained many-core system," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 21, no. 7, pp. 890–902, 2011.

[13] I. Heo, S. Park, J. Lee, and Y. Paek, "An asip approach for motion estimation reusing resources for h.264 intra prediction," in *SoC Design Conference (ISOCC), 2010 International*, pp. 186 –189, nov. 2010.

[14] S. Momcilovic, N. Roma, and L. Sousa, "An asip approach for adaptive avc motion estimation," in *Research in Microelectronics and Electronics Conference, 2007. PRIME 2007. Ph.D.*, pp. 165–168, 2007.

[15] H. K. Eun, S. J. Hwang, M. H. Sunwoo, Y.-H. Kim, and H. S. Kim, "Integer-pel motion estimation specific instructions and their hardware architecture for asip," in *Circuits and Systems (ISCAS), 2011 IEEE International Symposium on*, pp. 953–956, 2011.

[16] H. C. Doan, H. Javaid, and S. Parameswaran, "Multi-asip based parallel and scalable implementation of motion estimation kernel for high definition videos," in *Embedded Systems for Real-Time Multimedia (ESTIMedia), 2011 9th IEEE Symposium on*, pp. 56–65, 2011.

[17] "Tensilica." Tensilica Inc. (http://www.tensilica.com).

[18] "H.264 reference software.." http://iphome.hhi.de/suehring/tml/.

[19] Wikipedia, "TILEPro64 Processor." http://en.wikipedia.org/wiki/TILEPro64.

[20] H. Javaid, M. Shafique, J. Henkel, and S. Parameswaran, "System-level application-aware dynamic power management in adaptive pipelined mpsocs for multimedia," in *Computer-Aided Design (ICCAD), 2011 IEEE/ACM International Conference on*, pp. 616–623, 2011.