A Case for FPGA-Based Accelerators for Energy-Efficient Motion Picture Video Processing

Jason Boyle and Shanker Shreejith

Trinity College Dublin, Dublin, Ireland

ABSTRACT

In recent years, computing-based video processing workflows have become an integral part of the motion picture industry. These workloads are highly data- and compute-intensive, requiring capable hardware to achieve the required performance. Currently, general-purpose CPUs and GPUs are used to accelerate video processing functions in motion picture workflows. While such devices are highly suited to software-driven video processing workflows, they consume large amounts of energy in these tasks. In this work, we present a case for deploying an FPGA-based accelerator as an energy-efficient alternative to general-purpose hardware in high-resolution motion picture video processing, using an ingest module as a case study. We show that an FPGA-based accelerator for decoding 8K OpenEXR B44 video frames, designed using a commercial high-level synthesis workflow and executing on a PCIe-connected Alveo U50 device, outperforms a highly parallel, CPU-optimised inbuilt B44 decoder implementation in terms of energy consumption per frame decoded. In our experiments, the FPGA-accelerated B44 was able to decode 8K frames with 47.9 ms latency while consuming 0.98 J of energy per frame, compared to the 58.3 ms achieved by a high-end Intel 11700-K CPU while consuming 4.5 J per frame, when averaged over 1000 runs. We further show that this offload can be seamlessly integrated into state-of-the-art motion picture tools such as NUKE with minimal effort. With FPGAs becoming mainstream in cloud servers, we envision that this work paves the way for more efficient integration and utilisation of custom hardware and FPGAs in compute-intensive motion picture workflows.

Keywords: Field-programmable gate arrays, Energy-efficient processing, Hardware-accelerators, High-performance computing, Motion pictures, Video processing

1. INTRODUCTION

Due to the massive popularity of Computer-Generated Imagery (CGI) and Visual Effects (VFX) in modern motion pictures, computing-based video processing workflows such as post-production and virtual production have become an integral part of the motion picture industry.[1] The compute- and data-intensive nature of these workloads demand the use of powerful hardware to achieve the desired level of visual quality and to minimise processing time. Currently, general-purpose Central Processing Units (CPUs) and Graphics Processing Units (GPUs) are used to accelerate motion picture video processing workflows.[2] These devices are often deployed in specialised server environments such as render farms, to enable offloading of compute-intensive video processing tasks on a large scale. In the motion picture industry, CPUs and GPUs are the hardware of choice, as these devices are highly optimised for software-driven video processing workloads, are natively supported by industry-standard video processing software, and can be easily deployed in server environments.[3] However, these devices consume large amounts of energy when executing compute-intensive workloads such as those used in the motion picture industry. This is exacerbated in large-scale environments such as render farms, incurring high costs and high environmental impact.[4]

Recent years have seen an increased interest in the use of dedicated hardware acceleration across the media processing pipeline, from sensors and image acquisition[5] to data- and compute-intensive post-production tasks.[6,7] In many application domains, dedicated hardware accelerators have been shown to achieve significant gains in performance and/or energy consumption, compared to commercial hardware.[8–11] Despite seeing increased

Further author information:
J.B.: E-mail: boylej6@tcd.ie
S.S.: E-mail: shankers@tcd.ie

adoption in video streaming applications,[11] dedicated hardware accelerators have seen limited usage in the motion picture industry. This can be attributed to the inherent challenges of designing efficient custom hardware with comparable or better performance than off-the-shelf solutions, requiring expertise in hardware design and implementation. Furthermore, seamlessly integrating a dedicated (custom) hardware accelerator into a motion picture video processing workflow poses another challenge, as the software applications used in the motion picture industry are designed and optimised for general-purpose hardware, and do not natively support custom hardware accelerators.

However, recent developments in hardware platforms and design tools are making the design and deployment of dedicated hardware accelerators more accessible. Field Programmable Gate Array (FPGA) hardware platforms such as the AMD Alveo family provide logic-dense FPGAs for implementing complex accelerator designs, paired with high-bandwidth global memory, and high-speed PCIe and networking interfaces, enabling high performance and easy integration into server environments.[12] The High-Level Synthesis (HLS) design flow enables highly optimised hardware accelerators to be developed at a high level of abstraction, through familiar programming languages such as C.[13] In addition, software Application Programming Interfaces (APIs) such as the AMD Xilinx RunTime (XRT) allow for seamless offloading to accelerators using function calls in software. In this work, we present the case for deploying an FPGA-based accelerator as a more energy-efficient alternative to general-purpose hardware in high-resolution motion picture video processing. The contributions of this work are as follows:

1. A highly parallel FPGA-based accelerator for decoding 8K (7680x4320) OpenEXR B44 video frames, designed using the AMD Vitis HLS design flow, and deployed on a PCIe-attached AMD Alveo U50 FPGA device. The accelerator features a number of optimisations during the design and implementation stages to maximise decoding performance and minimise energy consumption.

2. An FPGA-accelerated build of the OpenEXR software library, which facilitates seamless offloading of B44 decoding from the host system to the accelerator. The FPGA-accelerated OpenEXR library is highly portable, and can be used as a standalone B44 decoder, or can be easily integrated into state-of-the-art motion picture video processing software such as NUKE.[14]

3. An extensive analysis of the decoding performance and energy consumption of the FPGA-based accelerator, compared to a highly CPU-optimised inbuilt B44 decoder, run on a high-end Intel i7 11700K CPU.

OpenEXR B44[15] decoding was chosen as a case study, as OpenEXR B44 has seen increased usage in the motion picture industry in recent years, but lacks an efficient acceleration platform. The B44 decoding algorithm is arithmetically simple, but features a high degree of data dependency, limiting the number of parallel operations that can be performed. As a result, B44 decoding is currently accelerated using CPUs, since the limited parallelism hinders efficient GPU utilisation. While the OpenEXR inbuilt B44 decoder is highly optimised for CPU multithreading, the sequential processing nature of CPUs limits performance at high resolutions such as 8K. In this work, we leverage the inherent flexibility of dedicated hardware to design a highly efficient seven-stage pipeline to extract fine-grained parallelism from the B44 decoding algorithm. We show that the FPGA-accelerated B44 decoder can achieve 18% lower decoding latency and 4.6× lower energy consumption per 8K frame decoded in comparison to the OpenEXR CPU-optimised inbuilt B44 decoder running on a high-end Intel CPU.

2. BACKGROUND AND RELATED WORKS

The amount of data generated by commercial video production has vastly increased in the last decade, with several Terabytes generated per day by many commercial films.[16] Most modern production methods use multiple high-resolution cameras, sensors and CGI models to generate different scenes and sequences of the story. Post-production pipelines use sequences of read, processing and write operations multiple times to generate the final composited video. Complex tasks within this pipeline have been a prime target for acceleration on commercial compute accelerators such as GPUs, custom architectures on silicon (e.g., AV1 encoder/decoder) and reconfigurable platforms such as FPGAs (e.g., RED Rocket, Apple Afterburner).

Read/Write operations (Input/Output) tasks within the pipeline apply decompression/compression on the video frames to reduce the bandwidth required for transmission and/or storage. Modern GPUs integrate hardware accelerators to enable video stream encoding and decoding more efficiently with lower processing latency. Complex processing tasks within the pipeline such as 3D rendering, modelling, reconstruction and compositing also take advantage of the massive parallel processing offered by modern GPUs.[16,17] Specialised tasks such as texture synthesis[18] and feature point tracking,[19] among others, have also shown that GPU acceleration can improve real-time performance in the workflow.

Increasingly, deep-learning-based methods are also finding their way into GPU-accelerated post-production workflows. Denoising tasks that use deep-learning models are deployed across multiple stages, from in-camera noise reduction to VFX, and even in post-transcoding stages.[20–22] The authors in[23] show the use of a deep-learning-based scheme that uses an encoder-decoder architecture with a novel feature aggregation module to enable highly accurate video matting. Per-frame tasks such as colourisation[24,25] and chroma-keying[26] have also shown to benefit from GPU acceleration. While deep-learning inference can be performed on modern CPUs with many processing cores, the massively parallel architecture on GPUs and library support for mapping applications to these cores enables GPU to achieve superior acceleration in such data-independent tasks.

Researchers have also shown the potential benefits of using custom accelerators on FPGAs for such data-independent tasks. FPGA-based accelerators real-time for chroma-keying[7] stream the video input through the FPGA via a HDMI interface. FPGA-based accelerators for efficient caching of ray-tracing[27,28] and motion estimators have shown that they can achieve significantly better energy efficiency than CPU/GPU implementations. FPGAs are also an increasingly common resource in datacentres and cloud providers, enabling users to leverage heterogeneous acceleration workflows combining CPUs, GPUs and custom accelerators. With their custom levels of parallelism, FPGAs have also shown great promise as an energy-efficient acceleration platform for deep learning inference, reducing the requirements for additional logic through the tailored design of neural-network oriented accelerators and algorithmic optimisations that make deep learning computations FPGA friendly.[29] Additionally, the highly flexible nature of FPGAs enable the design of highly efficient pipelines to exploit the fine-grained parallelism within data-dependent tasks, allowing such tasks to be implemented more efficiently on an FPGA. However, the challenge remains that such designs require expertise in the design and optimisation of the accelerators' micro-architecture, and lack seamless integration into client-server workflows used in post-production.

2.1 Enabling acceleration at scale with FPGAs

To enable the deployment of custom hardware accelerators in massively parallel environments (like datacentres), FPGA vendors have developed specialised FPGA device families. The AMD Alveo family of data centre accelerator cards are aimed at accelerating compute-intensive workloads such as video processing, financial computing and cryptography in a client-server setting. The Alveo U50, whose architecture is shown in Figure 1, is a low-power, small form-factor card with a logic-dense Ultrascale+ FPGA fabric and 8GB of high-bandwidth memory (HBM) on the same die. The U50 also integrates a PCIe interface (gen3x16) and a 100 GbE network interface for data and control exchange to the host system. The FPGA fabric is partitioned into a user-accessible (dynamic) region for deploying custom accelerators and a static region that integrates the host interface (PCIe endpoints, direct memory access engines and execution scheduler). The HBM acts as the global memory for the user logic, interfaced via 32×AXI4 channels. The Alveo family also provides drivers and a runtime library (called XRT) for interfacing and managing the user-defined accelerator and the host interface primitives. The XRT APIs can be used to load accelerator designs onto the Alveo device, move data between the host system and the accelerator, and schedule accelerator execution. These API function calls can be integrated into a software workflow to offload compute-intensive tasks to the specialised accelerator(s) on the Alveo device. XRT also provides lightweight performance profiling by monitoring data transfer and accelerator execution times, and power profiling by monitoring the power draw from the PCIe power rails on the device. In this work, we use the Alveo U50 as a platform to show that a custom accelerator (a B44 decoding accelerator in our case) can be designed efficiently and integrated seamlessly with state-of-the-art post-production tools such as NUKE (through XRT), to improve performance and energy efficiency compared to the existing CPU-optimised OpenEXR B44 decoder.

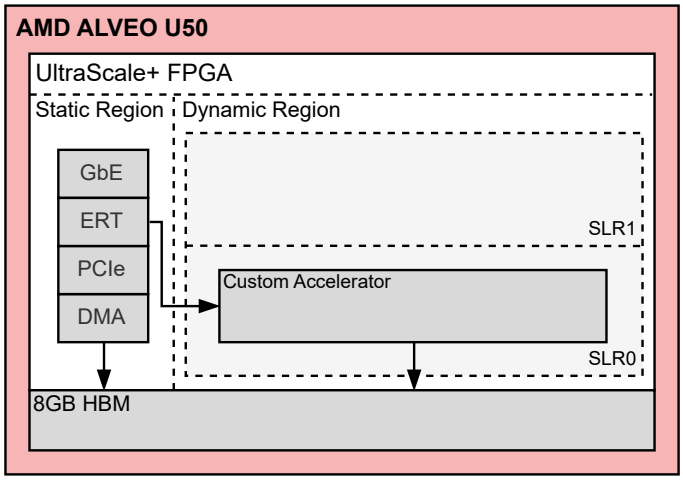

Figure 1. Simplified architecture of the AMD Alveo U50 data centre accelerator card.

2.2 OpenEXR B44

OpenEXR is a professional-grade image storage format, designed to accurately and efficiently store high-resolution, high dynamic range (HDR) images. It is used extensively in the motion picture industry and is the industry standard image storage format for VFX workflows. OpenEXR is implemented as an open-source C++ library,[30] making it highly portable and optimised for CPU acceleration in industry-standard VFX software applications such as NUKE. The OpenEXR standard supports a number of image compression methods, each designed for specific motion picture video processing workflows. Within the OpenEXR standard, B44 is a lossy compression method, designed for high decoding performance and maximum image quality, making it suitable for applications which require real-time playback.

B44 compresses channels containing 16-bit floating-point data, with a compression ratio of 2.28:1, or 44%. The size of a B44-compressed image is dependent only on the resolution of the image and the number of colour channels and is independent of the content of the pixels. Thus, all images of the same resolution and same number of colour channels are compressed to the same size. During B44 compression, each channel in the image is compressed independently. Each channel is divided into blocks of 4x4 pixels (32 bytes), which are compressed into 14-byte packets using the following algorithm:

1. The 16-bit floating-point pixel values are re-interpreted as 16-bit unsigned integers.
2. A set of fifteen difference values are computed by subtracting the values of adjacent pixels, as shown in Figure 2.
3. Each of the fifteen difference values are right-shifted according to a calculated shift value, and truncated to occupy 6 bits.
4. The 14-byte compressed packet is then assembled as shown in Figure 4.

When decoding a B44-compressed image, for each 14-byte packet, the 4x4-pixel block is unpacked by left-shifting each difference value according to the stored shift value, adding the result to the value of the adjacent pixel, and re-interpreting the 16-bit unsigned integer result as floating point. The decoded 4x4 blocks can then be assembled to re-construct each channel in the image. As B44 compresses an image by storing the differences between adjacent pixels, there is an inherent data dependency in the B44 decoding algorithm, as each pixel value can not be decoded until the adjacent pixel has been decoded. Figure 3 shows the data dependency graph between the pixel values in a 14-byte compressed packet. This data dependency prevents all sixteen pixels in the 4x4 block from being decoded in parallel. This data dependency is exploited by the CPU-optimised B44

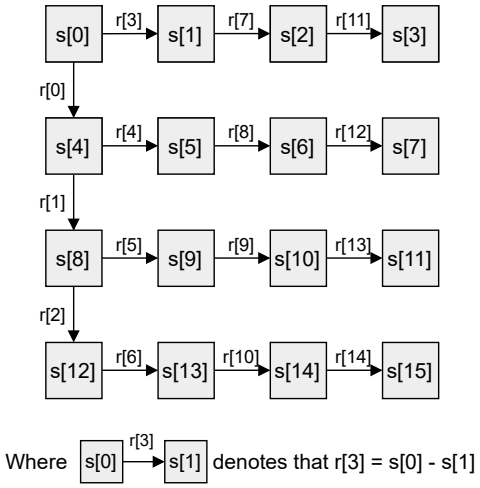

Figure 2. Calculation of difference values between adjacent pixels during B44 compression. The 16-bit pixel values are denoted by s[i]. The difference values are denoted by r[i].

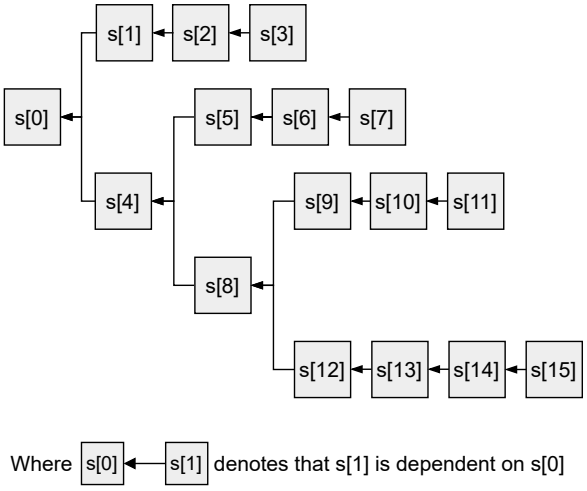

Figure 3. A graph of the data dependency between adjacent pixel values in a B44-compressed packet.

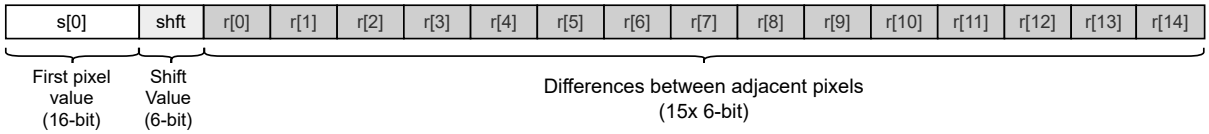

Figure 4. A 14-byte B44-compressed packet, containing a 4x4-pixel block.

decoder in OpenEXR's inbuilt implementation; however, the sequential operation limits the performance and energy efficiency at high resolutions such as 8K, even on heavily multithreaded CPUs.

In this work, we develop an FPGA implementation of the B44 decoding algorithm, targeting the AMD Alveo U50 device, which can be subsequently invoked from NUKE. By tailoring the design of the accelerator's datapath, we show that the low-level optimisations allow our approach to extract maximum performance from the decding algorithm, while minimising energy consumption.

3. ACCELERATOR DESIGN AND IMPLEMENTATION

3.1 System Overview

A high-level overview of the system is shown in Figure 5. The host system is a Linux server, equipped with an Intel Core i5 4770 CPU and 16GB of RAM. The Alveo U50 accelerator card is connected to the host system using a PCIe Gen3x16 interface, with the custom B44 decoding accelerator implemented within the dynamic region of the FPGA. The design and implementation of the B44 decoding accelerator are described in sections 3.2 and 3.3. The motion picture video processing software, NUKE, is run on the host system, with the encoded 8K OpenEXR B44 video frames stored in the host system memory. Within NUKE, EXR images are imported and decoded using a custom, FPGA-accelerated build of the OpenEXR library. Internally, the library uses XRT function calls to communicate with the B44 decoding accelerator. The FPGA-accelerated OpenEXR library is described in section 3.4. During runtime, when 8K OpenEXR B44 video frames are imported into NUKE, the FPGA-accelerated OpenEXR library seamlessly offloads the decoding of each frame to the dedicated accelerator. The accelerator operates by reading the encoded video frame from the host memory to the accelerator's global memory over PCIe, executing the B44 decode computation, and writing the decoded video frame from global memory to the host memory over PCIe. This is repeated for each frame in the video sequence.

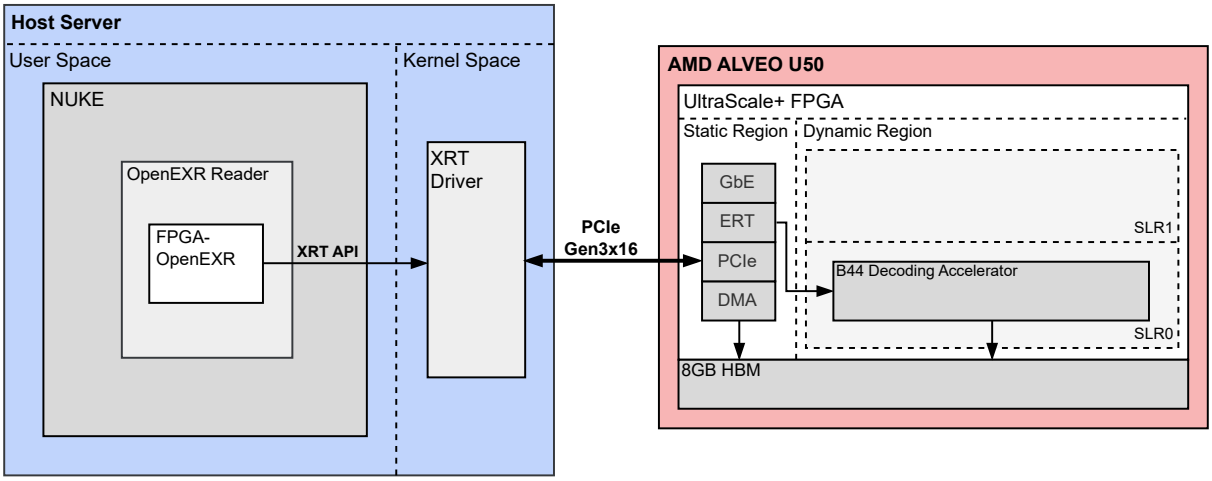

Figure 5. Overview of the system.

3.2 Accelerator Design

As mentioned, the B44 decoding accelerator is designed to decode a single 8K (7680x4320), OpenEXR B44 video frame with each execution. To maximise decoding throughput, the accelerator is made up of multiple B44 decode kernels, enabling multiple segments of the video frame to be decoded in parallel. All B44 decode kernels are identical and are designed to decode one segment from one channel of the video frame. To explore the trade-offs between performance, FPGA resource utilisation and power consumption, four configurations of the accelerator were developed and deployed, with each configuration leveraging a greater number of parallel B44 decode kernels to decode a greater number of frame segments in parallel. The accelerator configurations are referred to as B44Accel-n, where n denotes the number of B44 decode kernels within the accelerator. These accelerator configurations are summarised in Table 1. The minimal configuration, B44Accel-3, consists of three B44 decode kernels, one per colour channel, with each kernel responsible for decoding one full 7680x4320 channel. The maximal configuration, B44Accel-15, consists of fifteen B44 decode kernels, five per colour channel, with each responsible for decoding one 7680x864 segment from one channel.

In this work, we utilise the AMD Vitis C High-Level Synthesis (HLS) design flow to develop the B44 decode kernels. HLS enables rapid development and deployment of FPGA-based accelerators, by allowing the hardware to be developed at a high level of abstraction. By separating out memory access operations and computations into separate functions in high-level C language, HLS tools will infer and implement them as independent functional units. HLS pragmas are added to explicitly map specific aspects of a kernel such as function arguments and arrays to hardware blocks like interfaces and memory (blockRAM), allowing optimisations such pipelining to be applied at the compilation phase. For each configuration of the B44 decoding accelerator, only one B44 decode HLS kernel is developed, as kernels can be replicated to achieve the required level of parallelism during the implementation flow.

Table 1. Summary of the B44Accel-n configurations.

Name	Total Kernels	Kernels per Channel	Pixels Decoded per Kernel
B44Accel-3	3	1	7680x4320
B44Accel-6	6	2	7680x2160
B44Accel-12	12	4	7680x1080
B44Accel-15	15	5	7680x864

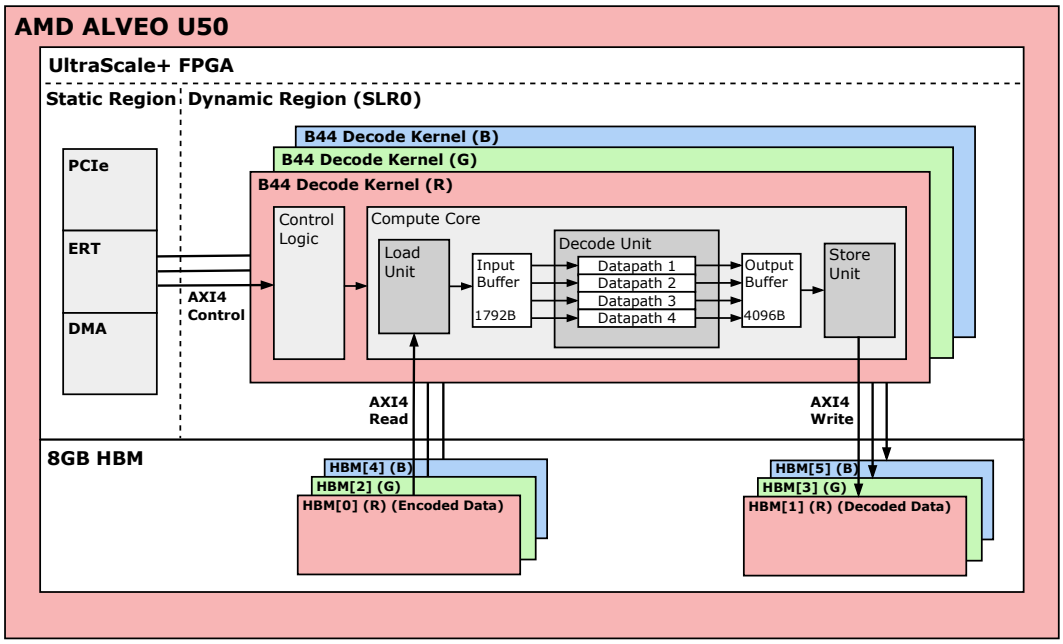

Figure 6. Micro-architecture of a B44 decode compute kernel. The B44Accel-3 accelerator is shown, which contains three such kernels, each designed to decode a full 7680x4320-pixel channel from the video frame.

The architecture of a B44 decode kernel is shown in Figure 6. The kernel contains control logic, which is used to manage the execution of the kernel, and a compute core, which performs the B44 decode computation. The control logic is automatically generated by the Vitis HLS flow based on the execution pipeline of the kernel. An AXI4 interface connects the control logic to the Embedded RunTime (ERT) scheduler in the static region. The compute core is partitioned into three main functional units: the load unit, the decode unit and the store unit. The compute core also contains local data buffers for on-chip storage of encoded and decoded data. The load and store units are each allocated a dedicated AXI4 data interface and HBM bank within global memory to enable independent read and write accesses to global memory. The AXI4 data interfacs are configured using HLS pragmas to enable burst transfers to maximise data throughput between the accelerator and global memory.

To maximise the decoding throughput of the kernel, the compute core is designed to execute using a three-stage load, decode, store pipeline. With each iteration of the load, decode, and store pipeline, a 512x4-pixel chunk (128 14-byte compressed packets) of image data is decoded. A chunk size of 512x4 was chosen, as this was found to offer the greatest balance between performance and FPGA resource utilisation. During the load stage, the load unit transfers 128 14-byte compressed packets (1792 bytes) from the input HBM bank to the local input buffer, using a single AXI4 burst transfer. During the decode stage, the decode unit utilises four parallel datapaths to read 14-byte compressed packets from the local input buffer, perform the B44 decode computation to unpack the 4x4-pixel blocks, and write the decoded pixel data into the local output buffer. The decode datapaths implement a custom seven-stage pipeline, designed to leverage the fine-grained parallelism in the B44 decoding algorithm. As mentioned in Section 2.2, the inherent data dependency in the B44 decoding algorithm prevents all sixteen pixels from being decoded in parallel. However, by grouping pixels based on shared dependencies, as shown in Figure 7, seven groups of pixels can be decoded in parallel. The seven-stage pipeline is designed to decode one such group of pixels in parallel during each stage to exploit this fine-grained parallelism. Four such datapaths in parallel exploits the data independence across the four rows of pixels in the chunk. During the store stage, the store unit transfers the decoded 512x4-pixel chunk (4096 bytes) from the local output buffer to the output HBM bank using four AXI4 burst transfers (of length 1024 bytes), one per row of pixels. The p ixels are stored in global memory according to their exact position within the image segment. This allows the full decoded image to be transferred directly from global memory to the frame buffer within the host memory, without requiring any data alignment by the host processor.

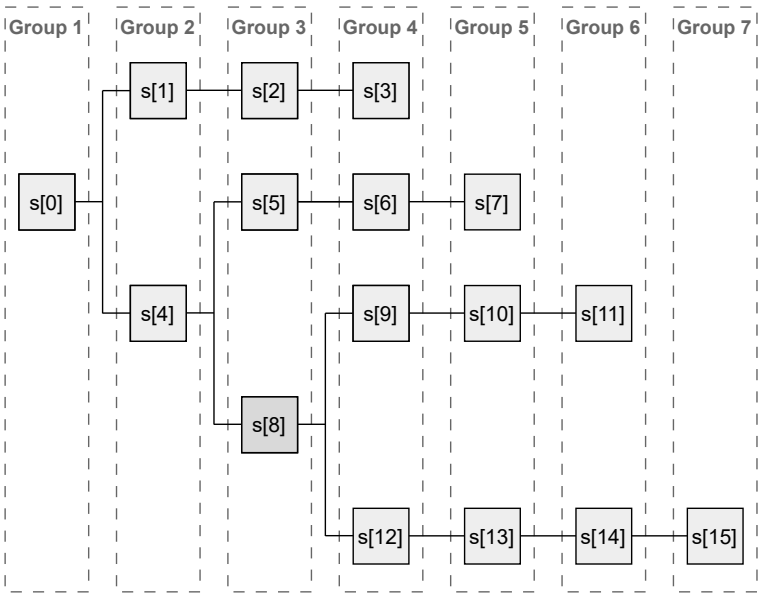

Figure 7. Illustration capturing groups of pixels in a B44-compressed packet which share common dependencies. The shared dependency enables these pixels to be decoded in parallel.

3.3 Accelerator Implementation

With the C HLS kernel developed, the Vitis HLS compiler is invoked to transform the C kernel into a synthesisable RTL IP block. Once generated, the RTL kernel is duplicated to the required number of kernels, which are then mapped to the Alveo U50 platform. During mapping, the physical location of each kernel is selected, and HBM banks are assigned to each kernel. In this design, we implement all kernels within Super Logic Region 0 (SLR0) of the FPGA, which contains the PCIe interface logic and the HBM interface logic to ensure the highest performance. Following the mapping stage, the standard FPGA flow is used to synthesise and implement the kernels and generate the bitstream file for programming the design onto the Alveo U50 platform.

3.4 Software Integration

All communication between the host system and the B44 decoding accelerator on the Alveo U50 are implemented within a custom build of the OpenEXR C++ library, referred to as FPGA-OpenEXR. Within the FPGA-OpenEXR library, XRT API calls are used to program the Alveo U50 with the accelerator's bitstream file, move image data between the host buffer and the accelerator, and schedule the kernels for parallel execution. Integrating the API calls within the library allows it to be compiled as a native standalone executable (using standard C++ compiler), or integrated into motion picture software tools such as NUKE. For the latter, the native OpenEXR reader node within NUKE is recompiled using the NUKE Developer Kit (NDK), and linked with the FPGA-OpenEXR library during compilation. Once compiled, any invocation of the OpenEXR reader node within NUKE seamlessly executes the decoding task on the accelerator on the attached Alveo U50 device.

4. EVALUATION

In this section, we evaluate the decoding latency and power consumption of each configuration of the B44 decoding accelerator and compare this to OpenEXR's highly CPU-optimised inbuilt B44 decoder. Each configuration of the B44 decoding accelerator is deployed on the Alveo U50 FPGA device, with an operating frequency of 300MHz. The host system in which the Alveo U50 is installed is described in Section 3.1. The OpenEXR inbuilt B44 decoder is run on a workstation equipped with an Intel Core i7 11700K CPU, operating at its base clock frequency of 3.6GHz. During execution on the CPU, multi-threading is enabled, allowing the i7 11700K to leverage all 16 threads when decoding each video frame.

For each configuration of the B44 decoding accelerator, decoding latency is measured as the total time for the accelerator to read the encoded video frame from host memory to global memory over PCIe, complete the decode computation, and write the decoded video frame from global memory to host memory over PCIe. For the CPU, decoding latency is measured as the wall-clock execution time of the multithreaded B44 decode computation. For each configuration of the B44 decoding accelerator, power consumption is measured using XRT's inbuilt power profiling, which reports the power draw from the 12V rail used to power the FPGA, and the 3.3V rail used to power the HBM. For the CPU, power consumption is measured using the Running Average Power Limit (RAPL) interface. These measurements include the power consumption of the CPU only, and do not include other devices such as system memory or storage. For both the B44 decoding accelerator and the CPU, decoding latency and power consumption measurements are averaged over 1000 8K decodes.

4.1 Decoding Latency and Throughput

Figures 8 and 9 show the decoding latency per 8K frame decoded, and the overall decoding throughput of each configuration of the B44 decoding accelerator and the OpenEXR inbuilt B44 decoder. It can be observed that the B44Accel-15 configuration achieves the overall lowest decoding latency of 47.87ms, and greatest throughput of 20.89 frames per second (fps), outperforming the OpenEXR inbuilt B44 decoder running on the 16-thread Intel 11700K by ∼18% in terms of decoding latency and ∼22% in terms of throughput. The decoding latency of each configuration of the B44 decoding accelerator is further broken down in Table 2. Across the four accelerator configurations, it can be seen that the decode computation time decreases linearly as the number of compute kernels within the accelerator increases, due to the greater level of parallelism. Excluding PCIe transfer latency, the B44Accel-12 and B44Accel-15 accelerator configurations significantly outperform the OpenEXR inbuilt B44 decoder in terms of decode computation latency alone. However, the overhead of transferring data to and from the accelerator's global memory significantly adds to the overall latency of the accelerator. In the B44Accel-15 configuration, PCIe transfer latency makes up ∼50% of the overall latency.

4.2 Resource Utilisation

The resource utilisation of each configuration of the B44 decoding accelerator is reported by the Vitis tool, following the implementation flow targeting the Alveo U50 FPGA device. Table 3 shows a detailed breakdown of the FPGA resources consumed by each accelerator configuration. In all configurations, the shell, which implements the essential elements of the platform such as the PCIe endpoint, DMA engine and runtime kernel scheduler, consumes a greater number of FPGA resources than the compute kernels. The highest performing configuration in terms of decoding latency and throughput, B44Accel-15, consumes a total of 35.54% and 18.94% of the available combinational Look-Up Table (LUT) and sequential register (REG) resources respectively, leaving ample

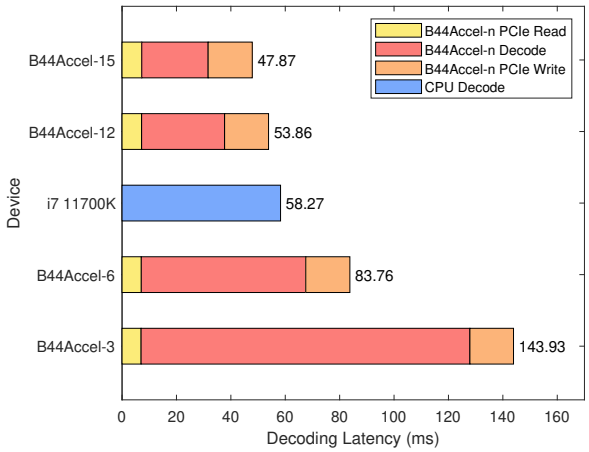

Figure 8. B44 decoding latency per 8K video frame, averaged over 1000 decoding cycles.

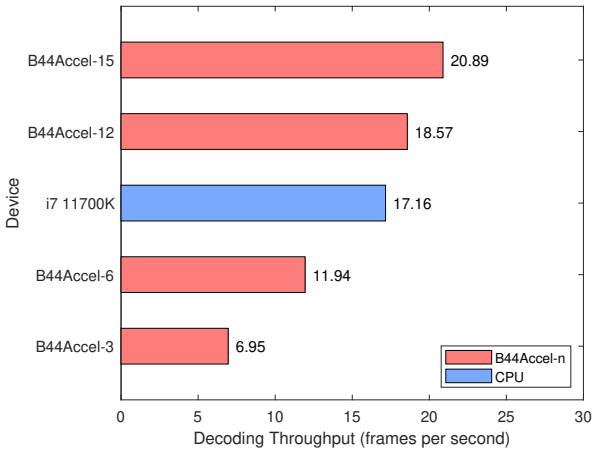

Figure 9. B44 decoding throughput for 8K video frames, averaged over 1000 decoding cycles.

Table 2. Detailed breakdown of the average decoding latency of each B44Accel-n configuration, showing the PCIe read latency, the decode computation latency and the PCIe write latency.

Name	Accelerator PCIe read latency (ms)	Accelerator Kernel decode latency (ms)	Accelerator PCIe write latency (ms)	Total decode latency(ms)
B44Accel-3	7.00±0.02	120.89±0.04	16.03±0.31	143.93±0.06
B44Accel-6	7.07±0.02	60.47±0.05	16.16±0.10	83.76±0.05
B44Accel-12	7.19±0.04	30.51±0.06	16.22±0.03	53.86±0.11
B44Accel-15	7.24±0.12	24.41±0.05	16.23±0.26	47.87±0.30
i7 11700K	-	-	-	58.27±0.83

Table 3. FPGA resource utilisation of each configuration of the B44 decoding accelerator.

Name	Component	#LUT	#LUTRAM	#REG	#BRAM	#URAM	#DSP	#HBM
B44Accel-3	Shell	129156	10501	170732	181	4	4	0
	Accelerator	26286	4308	21050	3	0	0	6
	Total	155442	14809	191782	184	4	4	6
	(%)	18.37%	2.71%	11.13%	13.73%	0.62%	0.07%	18.75%
B44Accel-6	Shell	134361	11836	181111	181	4	4	0
	Accelerator	54686	8616	44346	6	0	0	12
	Total	189047	20452	225547	187	4	4	12
	(%)	22.82%	5.14%	13.23%	13.99%	0.62%	0.07%	37.50%
B44Accel-12	Shell	145469	14507	202304	183	4	4	0
	Accelerator	110492	17232	87710	12	0	0	24
	Total	255961	31739	290014	195	4	4	24
	(%)	31.95%	8.04%	17.29%	14.65%	0.62%	0.07%	75.00%
B44Accel-15	Shell	150655	15842	212700	183	4	4	0
	Accelerator	131304	21540	103120	15	0	0	30
	Total	281959	37382	3158209	198	4	4	30
	(%)	35.54%	9.50%	18.94%	14.91%	0.62%	0.07%	93.75%

resources to implement compute kernels for other video processing functions in a motion-picture workflow. Implementing compute kernels for multiple video processing functions, connected together in a pipelined structure, would create a highly efficient accelerator, as multiple video processing operations could be applied to each video frame in a single execution of the accelerator.

4.3 Power and Energy Consumption

Figure 10 shows the power consumption per 8K frame decoded for each configuration of the B44 decoding accelerator and the OpenEXR inbuilt B44 decoder, running on the Intel 11700K CPU. Table 4 further breaks down the power draw from the Alveo U50's PCIe 12V and 3.3V rails. It can be observed that all configurations of the B44 decoding accelerator consume significantly less power than the Intel 11700K, with the highest performing accelerator, B44Accel-15, consuming ∼3.8× less power per frame than the inbuilt B44 decoder. In addition to measuring power consumption, the energy consumption per frame decoded was computed by multiplying the decode latency per frame with the power consumption per frame. This is shown in Figure 11. In terms of energy consumption per frame decoded, it can be observed that all accelerator configurations are significantly more energy efficient than the OpenEXR inbuilt B44 decoder running on the Intel 11700K CPU. For the B44Accel-15 configuration, the lower decoding latency and power consumption make this accelerator ∼4.6× more energy efficient than the OpenEXR inbuilt B44 decoder running on the Intel i7 11700K. This demonstrates the high degree of performance and energy efficiency which can be achieved by offloading such tasks to a dedicated hardware accelerator.

Table 4. Power Consumption of each configuration of the FPGA-based accelerator

Name	PCIe 12V Rail Power Draw (W)	PCIe 3.3V Rail Power Draw (W)	Total Power Draw (W)
B44Accel3	14.73±0.04	0.83±0.01	15.56±0.05
B44Accel6	15.13±0.05	0.95±0.01	16.09±0.05
B44Accel12	18.10±0.07	1.76±0.01	19.86±0.08
B44Accel15	18.50±0.09	1.89±0.01	20.39±0.10
i7 11700K	-	-	77.43±1.81

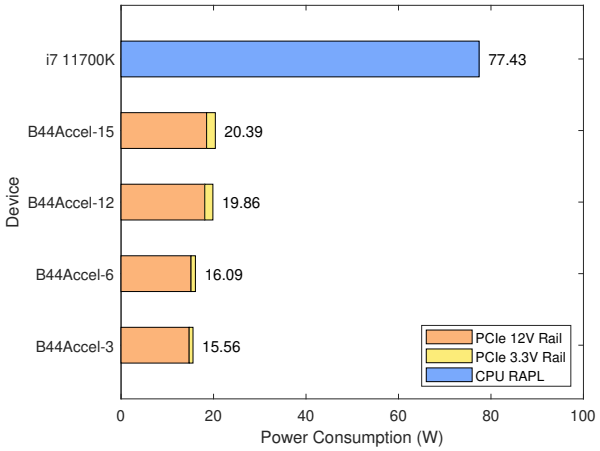

Figure 10. Power consumption per 8K video frame decoded, averaged over 1000 decode cycles.

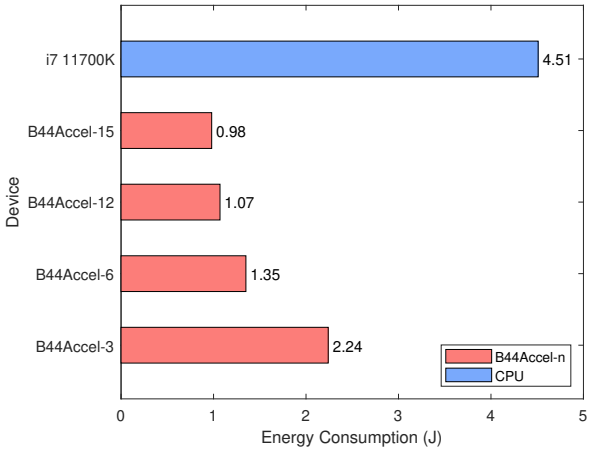

Figure 11. Energy consumption per 8K video frame decoded, averaged over 1000 decode cycles.

5. CONCLUSION

In this paper, we present a case for using custom-defined accelerators on FPGAs to offload data- and compute-intensive video processing tasks in a motion picture workflow. We use OpenEXR B44 decoding as a case study, to show that custom datapath optimisations can extract fine-grained parallelism within a data-dependent decoding algorithm, enabling superior decoding performance than general-purpose hardware. The implementation of a single-channel B44 decode kernel is designed using a high-level C representation, from which, highly-parallel, multi-kernel B44 decoding accelerators are generated through high-level synthesis, and deployed on a datacentre-class Alveo U50 FPGA device. Utilising the AMD XRT runtime library allows for seamless integration of the B44 decoding accelerator into the host's software environment, enabling the accelerator to be invoked from a standalone executable on the host, or integrated into state-of-the-art motion picture software such as NUKE. Our results show that the highest performing configuration of the B44 decoding accelerator can reduce the decoding latency of an 8K video frame by 18%, compared to OpenEXR's inbuilt, CPU-optimised B44 decoder running on a 16-threaded Intel i7 11700K CPU. We also show that offloading B44 decoding to the FPGA-based accelerator can reduce the energy consumption per frame decoded by up to 4.6× compared to the CPU, making it an ideal choice for decoding at scale in VFX and motion picture workflows.

ACKNOWLEDGMENTS

We would like to acknowledge many fruitful conversations with Dan Ring, Head of R&D at The Foundry as well as the support of Prof. Anil Kokaram of the Sigmedia Group at the EEE Dept., Trinity College Dublin, for making this publication possible

REFERENCES

[1] May, S., "How Pixar Sees Real Time Technologies Impacting Feature Animation Pipelines," in [*Real Time Conference 2021: Special Session*], (2021).

[2] True, T. and Sylvester-Bradley, G., "COTS (Commercial-off-the-Shelf) Platform for Media Production Everywhere," *SMPTE Motion Imaging Journal* **132**(2), 15–25 (2023).

[3] Kernen, T. and True, T., "Tighter NIC/GPU Integration Yields Next-Level Media Processing Performance," *SMPTE Motion Imaging Journal* **132**(1), 59–68 (2023).

[4] Helzle, V., Spielmann, S., and Trottnow, J., "Green screens, green pixels and green shooting," in [*ACM SIGGRAPH 2022 Talks*], 1–2 (2022).

[5] Theuwissen, A., Coghill, J., Ion, L., Shu, F., Siefken, H., and Smith, C., "Ultra-high resolution image capturing and processing for digital cinematography," in [*IEEE International Solid-State Circuits Conference, 2003. Digest of Technical Papers. ISSCC.*], 162–485 vol.1 (2003).

[6] Merah, H., Merah, L., Chaib, N., and Ali-Pacha, A., "Designing and Real-Time FPGA-Based Implementation of a Chroma-Key Effect System Using Xilinx System Generator," *International Journal of Future Computer and Communication* **11**(4) (2022).

[7] Sang, N. T. and Vinh, T. Q., "FPGA implementation for real-time Chroma-key effect using Coarse and Fine Filter," in [*International Conference on Computing, Management and Telecommunications (ComManTel)*], 157–162 (2013).

[8] Zhang, N., Yang, B., Chen, C., Yin, S., Wei, S., and Liu, L., "Highly efficient architecture of newhopenist on fpga using low-complexity ntt/intt," *IACR Transactions on Cryptographic Hardware and Embedded Systems* , 49–72 (2020).

[9] Dai, G., Huang, T., Chi, Y., Xu, N., Wang, Y., and Yang, H., "Foregraph: Exploring large-scale graph processing on multi-fpga architecture," in [*Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*], 217–226 (2017).

[10] You, Y., Zhang, Z., Hsieh, C.-J., Demmel, J., and Keutzer, K., "Fast deep neural network training on distributed systems and cloud tpus," *IEEE Transactions on Parallel and Distributed Systems* **30**(11), 2449–2462 (2019).

[11] Golson, C., Vos, K., Guiot, F., and Doyle, G., "MXF Practice and Reconfigurable Application Specific Computing," *SMPTE Motion Imaging Journal* **115**(7-8), 248–254 (2006).

[12] UG1120, *Alveo Data Center Accelerator Card Platforms User Guide* (2022).

[13] UG1139, *Vitis High-Level Synthesis User Guide* (2023).

[14] "NUKE — VFX and Film Editing Software." https://www.foundry.com/products/nuke-family/nuke.

[15] "OpenEXR." www.openexr.com.

[16] Blat, J., Evans, A., Agenjo, J., Kim, H., Imre, E., Hilton, A., Tefas, A., Nikolaidis, N., Pitas, I., Polok, L., et al., "IMPART: Big media data processing and analysis for film production," in [*IEEE International Conference on Multimedia & Expo Workshops (ICMEW)*], 1–5, IEEE (2015).

[17] Briand, G., Bidgolirad, F., Szlapka, J., Lavalou, J., Lanouiller, M., Christie, M., Lvoff, J., Bertolino, P., and Guillou, E., "On-set previsualization for vfx film production," (2014).

[18] Gallagher, C., Kelly, F., and Kokaram, A., "Fast scale invariant texture synthesis with GPU acceleration," in [*IEE European Conference on CVMP Visual Media Prodcution, London, UK*], (2005).

[19] Bailer, W., Fassold, H., Rosner, J., and Thallinger, G., "Innovative Tools for 3D Cinema Production,"

[20] Guo, S., Yan, Z., Zhang, K., Zuo, W., and Zhang, L., "Toward convolutional blind denoising of real photographs," in [*Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*], 1712–1722 (2019).

[21] Zamir, S. W., Arora, A., Khan, S., Hayat, M., Khan, F. S., Yang, M.-H., and Shao, L., "Multi-stage progressive image restoration," in [*Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*], 14821–14831 (2021).

[22] Liang, J., Cao, J., Sun, G., Zhang, K., Van Gool, L., and Timofte, R., "Swinir: Image restoration using swin transformer," in [*Proceedings of the IEEE/CVF international conference on computer vision*], 1833–1844 (2021).

[23] Sun, Y., Wang, G., Gu, Q., Tang, C.-K., and Tai, Y.-W., "Deep video matting via spatio-temporal alignment and aggregation," in [*Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*], 6975–6984 (2021).

[24] Górriz, M., Mrak, M., Smeaton, A. F., and O'Connor, N. E., "End-to-End Conditional GAN-based Architectures for Image Colourisation," in [*arXiv:1908.09873*], (2019).

[25] Blanch, M. G., Khalifeh, I., Smeaton, A., O'Connor, N., and Mrak, M., "Attention-based Stylisation for Exemplar Image Colourisation," in [*arXiv:2105.01705*], (2021).

[26] Yin, L. and Zhao, J., "Hybrid key: An automatic tool for real-time high quality chroma keying," in [*IEEE International Conference on Image Processing (ICIP)*], 4853–4857 (2015).

[27] Collinson, S. and Sinnen, O., "Caching architecture for flexible FPGA ray tracing platform," *Journal of Parallel and Distributed Computing* **104**, 61–72 (2017).

[28] Collinson, S., *Efficient Ray Tracing on FPGAs*, PhD thesis, ResearchSpace@ Auckland (2014).

[29] Guo, K., Zeng, S., Yu, J., Wang, Y., and Yang, H., "A survey of FPGA-based neural network accelerator," *arXiv preprint arXiv:1712.08934* (2017).

[30] Foundation, A. S., "OpenEXR Github." https://github.com/AcademySoftwareFoundation/openexr/tree/v2.4.1.