Warehouse-Scale Video Acceleration: Co-design and Deployment in the Wild

Parthasarathy Ranganathan
Daniel Stodolsky
Jeff Calow
Jeremy Dorfman
Marisabel Guevara
Clinton Wills Smullen IV
Aki Kuusela
Raghu Balasubramanian
Sandeep Bhatia
Prakash Chauhan
Anna Cheung
In Suk Chong
Niranjani Dasharathi
Jia Feng
Brian Fosco
Samuel Foss
Ben Gelb
Google Inc.
USA

Sarah J. Gwin
Yoshiaki Hase
Da-ke He
C. Richard Ho
Roy W. Huffman Jr.
Elisha Indupalli
Indira Jayaram
Poonacha Kongetira
Cho Mon Kyaw
Aaron Laursen
Yuan Li
Fong Lou
Kyle A. Lucke
JP Maaninen
Ramon Macias
Maire Mahony
David Alexander Munday
Srikanth Muroor
vcu@google.com
Google Inc.
USA

Narayana Penukonda
Eric Perkins-Argueta
Devin Persaud
Alex Ramirez
Ville-Mikko Rautio
Yolanda Ripley
Amir Salek
Sathish Sekar
Sergey N. Sokolov
Rob Springer
Don Stark
Mercedes Tan
Mark S. Wachsler
Andrew C. Walton
David A. Wickeraad
Alvin Wijaya
Hon Kwan Wu
Google Inc.
USA

ABSTRACT

Video sharing (e.g., YouTube, Vimeo, Facebook, TikTok) accounts for the majority of internet traffic, and video processing is also foundational to several other key workloads (video conferencing, virtual/augmented reality, cloud gaming, video in Internet-of-Things devices, etc.). The importance of these workloads motivates larger video processing infrastructures and – with the slowing of Moore's law – specialized hardware accelerators to deliver more computing at higher efficiencies. This paper describes the design and deployment, at scale, of a new accelerator targeted at warehouse-scale video transcoding. We present our hardware design including a new accelerator building block – the *video coding unit (VCU)* – and discuss key design trade-offs for balanced systems at data center scale and co-designing accelerators with large-scale distributed software systems. We evaluate these accelerators "in the wild" serving live data center jobs, demonstrating 20-33x improved efficiency over our prior well-tuned non-accelerated baseline. Our design also enables effective adaptation to changing bottlenecks and improved failure management, and new workload capabilities not otherwise possible with prior systems. To the best of our knowledge, this is the first work to discuss video acceleration at scale in large warehouse-scale environments.

ASPLOS '21, April 19–23, 2021, Virtual, USA

ACM ISBN 978-1-4503-8317-2/21/04.
https://doi.org/10.1145/3445814.3446723

CCS CONCEPTS

• **Hardware → Hardware-software codesign**; • **Computer systems organization → Special purpose systems**.

KEYWORDS

video transcoding, warehouse-scale computing, domain-specific accelerators, hardware-software codesign

ACM Reference Format:
Parthasarathy Ranganathan, Daniel Stodolsky, Jeff Calow, Jeremy Dorfman, Marisabel Guevara, Clinton Wills Smullen IV, Aki Kuusela, Raghu Balasubramanian, Sandeep Bhatia, Prakash Chauhan, Anna Cheung, In Suk Chong, Niranjani Dasharathi, Jia Feng, Brian Fosco, Samuel Foss, Ben Gelb, Sarah J. Gwin, Yoshiaki Hase, Da-ke He, C. Richard Ho, Roy W. Huffman Jr., Elisha Indupalli, Indira Jayaram, Poonacha Kongetira, Cho Mon Kyaw, Aaron Laursen, Yuan Li, Fong Lou, Kyle A. Lucke, JP Maaninen, Ramon Macias, Maire Mahony, David Alexander Munday, Srikanth Muroor, Narayana Penukonda, Eric Perkins-Argueta, Devin Persaud, Alex Ramirez, Ville-Mikko Rautio, Yolanda Ripley, Amir Salek, Sathish Sekar, Sergey N. Sokolov, Rob Springer, Don Stark, Mercedes Tan, Mark S. Wachsler, Andrew C. Walton, David A. Wickeraad, Alvin Wijaya, and Hon Kwan Wu. 2021. Warehouse-Scale Video Acceleration: Co-design and Deployment in the

Wild. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '21), April 19–23, 2021, Virtual, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3445814.3446723

1 INTRODUCTION

Video sharing services are vital in today's world, providing critical capabilities across the globe to education, business, entertainment and more. Video is the dominant form of internet traffic, making up >60% of global internet traffic as of 2019 [10], and continues to grow given 4K and 8K resolutions and emerging technologies such as augmented and virtual reality, cloud video gaming, and Internet-of-Things devices. Recently, the COVID-19 pandemic has further amplified the importance of internet video platforms for communication and collaboration: e.g., medical professionals using video platforms to share life-saving procedures or increased YouTube usage (>15% of global internet traffic) [11].

While the computational demand for video processing is exploding, improvements from Moore's Law have stalled [27]. Future growth in this important area is not sustainable without adopting domain-specific hardware accelerators. Prior work on video acceleration has focused primarily on consumer and end-user systems (e.g., mobile devices, desktops, televisions), with few video products targeting data centers [37]. Introducing video transcoding accelerators at warehouse-scale [4] is a challenging endeavor. In addition to the high quality, availability, throughput, and efficiency requirements of cloud deployments, the accelerator must support the complexity of server-side video transcoding (i.e., plethora of formats and complex algorithmic and modality trade-offs), deployment at scale (i.e., workload diversity and serving patterns), and co-design with large-scale distributed systems.

In this paper, we address these challenges. To the best of our knowledge, this is the first work to discuss the design and deployment of warehouse-scale video acceleration at scale in production. Specifically, we make the following key contributions.

First, we present a *new holistic system design for video acceleration, built ground up for warehouse-scale data centers*, with a new hardware accelerator building block – the *video coding unit (VCU)* – designed to work in large distributed clusters with warehouse-scale schedulers. We detail our carefully co-designed abstractions, partitioning, and coordination between hardware and software, as well as specific design and engineering optimizations at the levels of hardware blocks, boards, nodes, and geographically-distributed clusters. For example, VCUs implement a sophisticated acceleration pipeline and memory system, but are also designed with support for stateless operations and user-space programmability to work better with data center software. Similarly, our clusters are carefully optimized for system balance under increased diversity and density, but also support rich resource management abstractions and new algorithms for work scheduling, failure management, and dynamic tuning. Additionally, we discuss our approach to using high-level synthesis to design our hardware for deeper architecture evaluation and verification.

Second, we present *detailed data and insights from our deployment at scale in Google* including results from longitudinal studies across tens of thousands of servers. Our accelerator system has an order of magnitude performance-per-cost improvement (20x-33x) over our prior well-tuned baseline system with state-of-the-art CPUs while still meeting strict quality, throughput, latency, and cost requirements across a range of video workloads (video sharing, photos/video archival, live streaming, and cloud gaming). We also present results demonstrating how our holistic co-design allows for real-world failure management and agility to changing requirements, as well as enables new capabilities that were previously not possible (increased compression, live video applications, etc).

The rest of the paper is organized as follows. Section 2 provides background on why data center scale video transcoding is a challenging workload to accelerate. Section 3 discusses our system design and implementation, with specific focus on new insights around system balance and hardware-software co-design specific to video acceleration at warehouse-scale. Section 4 presents measurements from at-scale deployment in our production data centers, Section 5 discusses related work, and Section 6 concludes the paper.

2 WAREHOUSE-SCALE VIDEO PROCESSING

In this section, we discuss key aspects of warehouse-scale video processing platforms that make it challenging for hardware acceleration. We also describe how data center transcoding differs from consumer devices.

2.1 Video Transcoding: Workload Challenges

A Plethora of Output Files: Video sharing platforms like YouTube enable a user to upload a video they created, and lets others reliably view it on a variety of devices (e.g., desktop, TV, or mobile phone). The video sharing platform (Figure 1) includes computing and storage in data centers and streaming via a content-delivery network (CDN) [15]. In this paper, we focus on the former two data center components. Given the large range of screen sizes/resolutions, from 8K TVs down to low-resolution flip phones, most video platforms will convert each uploaded video into a standard group of 16:9 resolutions[1]. These video files are computed and saved to the cloud storage system and served as needed. This production of multiple outputs per input is a key difference between a video sharing service and a consumer video application like video chat.

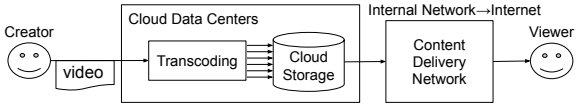


Figure 1: Video platform functional diagram

Since lower resolutions have smaller file sizes and can be upscaled on a viewer's device, the client may adapt to limited/changing bandwidth by requesting a lower resolution (e.g., adaptive bitrate or ABR [7, 16, 48]).

[1]For example, 256 x 144, 426 x 240, …, 3840 x 2160 (a.k.a. 4K), 7680 x 4320 (a.k.a. 8K)). These are usually shortened to just the vertical dimension (e.g. 144p, 240p, …, 2160p, and 4320p).

A Plethora of Video Formats: Compressing video files makes them much smaller, yielding storage and network bandwidth benefits. *Video coding specifications* define how to decompress a compressed video sequence back into pictures and *Codecs* are implementations of these specifications. Popular coding specifications include H.264/AVC [28], VP9 [21], and AV1 [12]. High compression is achieved using combinations of prediction, transformation, quantization, entropy coding, and motion estimation [22, 24, 55]. Newer specifications use more computation for higher compression gains.

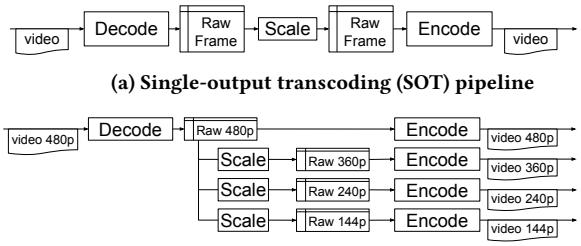
While some devices (laptops, desktops) keep up with the latest specifications via software decoders (running on general-purpose processors), others (TV, mobile) use hardware (fixed-function) decoders for their power efficiency and speed and thus continue to stick with older specifications. Therefore, to leverage new specifications when the viewer's device supports it and use older ones when the device does not, videos must be encoded in a plethora of different formats. Combined with the multiple resolutions described above, this translates to a majority of work in the video processing platform spent on *transcoding*. Contrast this with classic video broadcast (TV) where video is encoded in one format and resolution and all playback devices support that same format/resolution!

Algorithmic Trade-Offs in Video Transcoding: Figure 2a shows the *transcoding* process by which a video is decoded from one format into raw frames, scaled to the output resolution and then encoded into another, potentially different, format, typically with higher compression settings than the original consumer encoder. Video sharing platforms must optimize these trade-offs to ensure that users receive playable and high quality video bitstreams while minimizing their own computational and network costs.

Encoding is a computationally hard search problem often taking many orders-of-magnitude longer than decoding, involving trade-offs between perceptual quality, resultant bitrate, and required computation [55]. The encoder exploits redundancy within and across frames to represent the same content in fewer bytes. The high compute cost is due to the large search space of encoding parameters, which is a combination of the resolution, motion, and coding specification. New compression specifications grow the search space by providing additional tools that the encoder can apply to better express the redundancy in video content in fewer bits.

Another key parameter to improve video quality and/or bitrate is the use of non-causal information about the video frame sequence. This leads to a choice of *one-pass* or *two-pass* algorithms used in *low-latency*, *lagged*, or *offline* modes. The lowest-latency encoding (e.g., videoconferencing, gaming) is low-latency, one-pass encoding where each frame is encoded as soon as available but with limited information on how to allocate bits to frames. In two-pass encoding, frame complexity statistics are collected in the first pass and used to make frame type and bit allocation decisions in the second pass [61] over different time windows. Two-pass encoding can be additionally classified as below.

- Low-latency two-pass has no future information but is still able to use statistics from the current and prior frames to improve decisions on frame type and bit allocation.
- Lagged two-pass encoding has a window of statistics about future frames and allows for bounded latency (e.g., for live streams).
- The best quality is offline two-pass (e.g., used in large-scale video services like YouTube and Netflix) where frame statistics from the entire video are available when running the second pass.



(a) Single-output transcoding (SOT) pipeline

(b) 480p multiple-output transcoding (MOT) pipeline

Figure 2: Data center video transcoding patterns

Finally, advanced encoding systems [7, 33] may do multiple complete passes of any of the above encoding schemes combined with additional analysis (e.g., rate quality curves for individual videos at multiple operating points) to produce better quality/compression trade-offs at additional computational cost.

Chunking and Parallel Transcoding Modes: The video processing platform is designed to leverage warehouse infrastructure to run as much in parallel as possible. Transcoders can also shard the video into chunks (also known as closed Groups of Pictures, or GOPs) that can each be processed in parallel [17]. The transcoder can perform either *single-output transcoding* (SOT) or *multiple-output transcoding* (MOT). As shown in Figure 2a, SOT is a straightforward implementation of a transcoder service, simply reading an input chunk, decoding it, and then encoding a single output variant (possibly after scaling). A separate task must be used for each resolution and format desired.

MOT is an alternative approach where a single transcoding task produces the desired combination of resolutions and formats for a given chunk (Figure 2b). The input chunk is read and decoded once, and then downscaled and encoded to all output variants in parallel. This reduces the decoding overheads and allows efficient sharing of control parameters obtained by analysis of the source (e.g., detection of fades/flashes). MOT is generally preferred to SOT, as it avoids redundant decodes for the same group of outputs, but SOT may be used when memory or latency needs mandate it.

2.2 Warehouse-Scale Processing: Challenges

Multiple Video Workloads and Requirements: YouTube's video processing platform [7, 34] currently supports multiple video-centric workloads at Google: (1) YouTube itself that handles uploads of multiple hundreds of hours of video every minute, (2) Google Photos and Google Drive with a similar volume of videos, and (3) YouTube Live with hundreds of thousands of concurrent streams. These services differ in their load and access patterns. Their end-to-end latency requirements also vary widely, from Live's 100 ms to video upload's minutes to hours. As discussed above, spreading the

work across many data centers around the world helps distribute the load and meet latency requirements.

Video Usage Patterns at Scale: As with other internet media content [25], video popularity follows a stretched power law distribution, with three broad buckets. The first bucket – the *very popular* videos that make up the majority of watch time – represents a small fraction of transcoding and storage costs, worth spending extra processing time to reduce bandwidth to the user. The second bucket includes *modestly watched* videos which are served enough times to motivate a moderate amount of resources. And finally, the third bucket includes the *long tail*, the majority of videos that are watched infrequently enough that it makes sense to minimize storage and transcoding costs while maintaining playability. Note that old videos can increase in popularity and may need to be reprocessed with a higher popularity treatment well after upload.

Data Center Requirements: Designing video transcoding ASICs for the data center can be fundamentally different than designing for consumer devices. At the warehouse-scale, where many thousands of devices will be deployed, there is an increased focus on cost efficiency that translates into a focus on throughput and scale-out computing [4]. The "time to market" also becomes critical, as launching optimized products faster can deliver significant cost savings at scale. Additionally, unlike consumer environments where individual component reliability and a complete feature set are priorities, in a warehouse-scale context, the constraints are different: fallback software layers can provide infrequently needed features and reliability can be augmented by redundant deployments. Also, at large scale, testing and deploying updates can be highly disruptive in data centers, and consequently systems need to be optimized for change management.

Data Center Schedulers: One key characteristic of warehouse-scale designs is the use of a common software management and scheduling infrastructure across all computing nodes to orchestrate resource usage across multiple workloads (e.g., Google's Borg [59]). This means that the video processing platform is closely designed with the warehouse-scale scheduler. Processing starts with identifying what output variants need to be generated for a given video based on its characteristics and the application (video sharing, storage, streaming, etc.). Based on the required output variants, an acyclic task dependency graph is generated to capture the work to be performed. The graph is placed into a global work queue system, where each operation is a variable-sized "step" that is scheduled on machines in the data center to optimize available capacity and concurrency. The step scheduling system distributes the load, adapting to performance and load variations as well as service or infrastructure failures. The video system also orchestrates the parallelism from chunking discussed earlier: breaking the video into chunks, sending them to parallel transcoder worker services, and assembling the results into playable videos. These kinds of platforms also operate at a global scale and thus the platform is distributed across multiple data centers. A video is generally processed geographically close to the uploader but the global scheduler can send it further away when local capacity is unavailable.

3 SYSTEM DESIGN

Summarizing the discussion above, transcoding is the most important component of data center video platforms but poses unique challenges for hardware acceleration. These include being able to handle and scale to a number of different output resolutions and formats, as well as handling complex algorithmic trade-offs and quality/compression/computing compromises. These challenges are compounded by attributes of warehouse-scale system design: inter- and intra-task parallelism, high performance at low costs, ease of deployment when operating at scale, co-ordinated scheduling and failure tolerance. Taken together, cloud video workloads on warehouse-scale computers are very different from their consumer counterparts, presenting new infrastructure challenges around throughput, quality, efficiency, workload diversity, reliability, and agility.

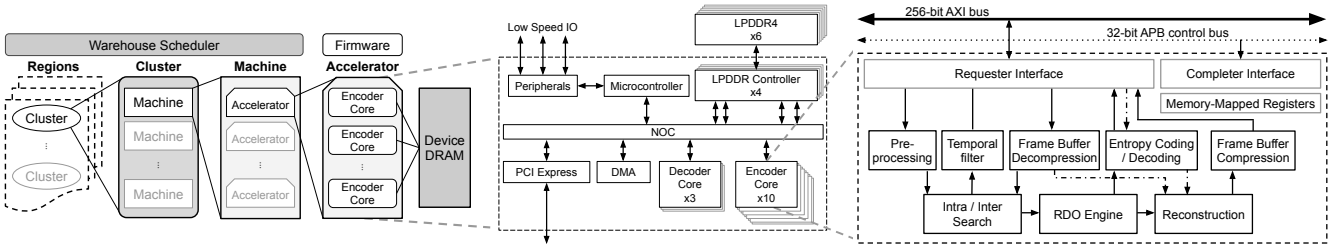
In response to these challenges, we designed a new holistic system for video acceleration, built ground-up for data-center-scale video workloads, with a new hardware accelerator building block – a *video coding unit (VCU)* – co-designed to work in large distributed clusters with warehouse-scale schedulers. Core to our solution is *hardware-software co-design*, to architect the system to scalably partition and optimize *functionality at individual levels* – from individual hardware blocks to boards, nodes, and geographically-distributed clusters, and across hardware, firmware, and distributed systems software – with appropriate *abstractions and interfaces between layers*. We follow a few key high-level design principles in optimizing for the distinct characteristics and constraints of a data center deployment:

Globally Maximize Utilization: Given power and die-area constraints are more relaxed, our data center ASICs are optimized for throughput and density, and multi-ASIC deployments amortize overheads. In addition, we optimize system balance and global work scheduling to minimize stranding (underutilized resources), specifically paying attention to the granularity and fungibility of work.

Optimize for Deployment at Scale: Software deployments have varying degrees of disruption in data centers: kernel and firmware updates require machine unavailability, in contrast to userspace deployments which only require, at most, worker unavailability. We therefore design our accelerators for userspace software control. Also, as discussed earlier, individual component reliability can be simplified at the warehouse level: hardware failures are addressed through redundancy and fallback at higher-level software layers.

Design for Agility and Adaptability: In addition to existing workload diversity, we have to plan for churn as applications and use-cases evolve over time. We therefore design programmability and interoperability in hardware, ossifying only the computationally expensive infrequently-changing aspects of the system. Software support is leveraged for dynamic tuning ("launch-and-iterate") as well as adapt to changing constraints. An emphasis on agility also motivates our use of high-level synthesis (HLS) to take a software-like approach to hardware design.

In the rest of this section, we describe how these principles translate to specific design decisions. Section 3.1 first introduces the holistically co-designed system. Section 3.2 discusses the design of our VCU hardware accelerator, and Section 3.3 discusses how the VCU and its system is co-designed to work in larger balanced



(a) Overview: globally-distributed clusters to chips **(b) VCU block diagram** **(c) Encoder core functional block diagram**

Figure 3: Design at all scales: global system, chip, and encoder core

clusters and with the firmware and distributed software stack. Section 3.4 discusses additional details of how we use HLS to accelerate our design, and Section 3.5 summarizes the design.

3.1 Video Accelerator Holistic Systems Design

Figure 3a shows our overall system design. Each cluster operates independently and has a number of VCU machines along with non-accelerated machines. Each VCU machine has – in addition to the host compute – multiple accelerator trays, each containing multiple VCU cards, which in turn contain multiple VCU ASICs. The VCU ASIC design is shown in Figure 3b and combines multiple encoder cores (discussed in Figure 3c) with *sufficient* decode cores, network-on-chip (NoC), and DRAM bandwidth to maintain encoder throughput and utilization across our range of use-cases (i.e., MOT, SOT, low-latency, offline two-pass).

At the *ASIC* level, we selected parts of transcoding to implement in silicon based on their maturity and computational cost. The encoding data path is the most expensive (in compute and DRAM bandwidth) and sufficiently stable that it was the primary candidate. After encode, decoding is highly stable and is the next most dominant compute cost, making it a natural second candidate. Much of the rest of the system is continuously evolving, from the encoding rate control software to work scheduling, so those areas were left flexible. Additionally, we created a firmware and software focused hardware abstraction that allowed for performance and quality improvements post-deployment that will be further discussed in Section 3.3.2.

At the *board* and *rack* levels, we chose to deploy multiple VCUs per host to amortize overheads and make it simpler to avoid stranding encoder throughput due to host resource exhaustion (i.e., VCU hosts only serve VCU workers). This was also done because a high density deployment fit our racking and data center deployment approaches better than augmenting every machine in a cluster with VCU capacity, allowing us to reuse existing hardware deployment and management systems.

At the *cluster* level, we augmented our video processing platform to account for the heterogeneous resources of the VCU in scheduling work. Our video processing platform schedules graphs of work from a cluster-wide work queue onto parallel worker nodes that includes both transcoding and non-transcoding steps. Each VCU worker node runs a process per transcode to constrain errors to a single step. This new work scheduler was fundamental to maximizing VCU utilization data center-wide, beyond just at the level of a single VCU. As most of the ASIC area consists of encoder cores, maximizing the encoder utilization is the key to maximizing VCU utilization. The decoder cores are also taken into consideration, as under-utilizing them leaves the host with unnecessary software decoding load. Multiple-output transcoding (MOT) was considered foundational for encoder utilization because of the benefits discussed in Section 2. The efficiency of decoding once, scaling, and encoding an entire MOT graph on a single VCU simplifies scheduling and reduces resource consumption at the data center level. The typical structure of a multi-output transcode is a single-decode and then the set of conventional 16:9 outputs (e.g. for 1080p inputs: 1080p, 720p, 480p, 360p, 240p, and 144p are encoded). This scales down the decode needs of the VCU by the number of outputs and generally only doubles the encoding requirements[2]. Few videos require an entire VCU for their MOT, so we designed our VCUs to perform multiple MOTs and SOTs in parallel to boost encoder and VCU utilization.

3.2 VCU Encoder Core Design

The encoder core (Figure 3c) is the main element of the VCU ASIC and is able to encode H.264 and VP9 while searching three reference frames. The core shares some architecture features with other prior published work [57] – pipelined architecture, local reference store for motion estimation and other state, acceleration of entropy encoding – but is optimized for data center quality, deployment, and power/performance/area targets.

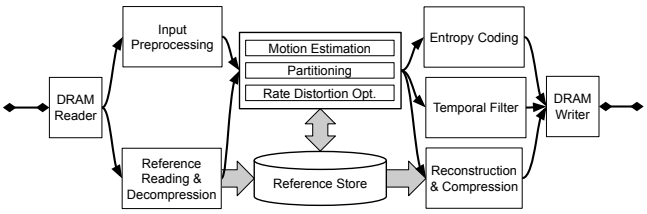


Figure 4: Encoder core functional pipeline

Figure 4 shows the main functional blocks in the pipeline (connected by small black arrows) as well as the data flow into and out of the reference store (connected by large gray arrows). The basic element of the pipelined computation is either a 16x16 *macroblock* (H.264) or a 64x64 *superblock*[3] (VP9) – the largest square group of pixels that a codec operates on at a time. Though the stages of the pipeline are balanced for expected throughput (cycles

[2]The pixel processing requirements of a multi-output transcode approximates a geometric series (e.g., 1080p is approximately 2 megapixels per frame; 720p + 480p + … + 144p sum to ~1.7 Mpixels).

[3]For simplicity, we will only talk about macroblocks in the rest of the discussion.

per macroblock), the wide variety of blocks and modes can lead to significant variability. To address this, the pipeline stages are decoupled with FIFOs, and full FIFO backpressure is used to stall upstream stages when needed.

Encoder Core Pipeline Stages: The first pipeline stage implements the classic stages of a block-based video encoding algorithm: *motion estimation*, *sub-block partitioning*, and *rate-distortion*-based transform and prediction mode selection [57, 65]. This is by far the most memory-bandwidth-intensive stage of the pipeline, interfacing heavily with the reference store (discussed below). A bounded recursive search algorithm is used for partitioning, balancing the coding overhead of smaller partitions against a reduction in net error. Per-codec logic selects from a number of transform/prediction mode candidates using approximate encoding/decoding to optimize bit rate and quality, and the number of rounds can be programmed. High-Level Synthesis (Section 3.4) was critical to experimenting with different algorithms and implementations.

The next stage implements *entropy encoding* for the output block, decoding of the macroblock (needed for the next stage), as well as *temporal filtering* for creating of VP9's alternate reference frames. This stage is sequential-logic-heavy and consequently challenging to implement in hardware [45]. While entropy decoding is fully defined by the specification, entropy encoding has many different algorithm and implementation options, e.g. VP9's per-frame probability adaptation [42]. Temporal filtering is a great example of an optimization that we added given the more relaxed die-area constraints in a data center use case. It uses motion estimation to align 16x16 pixel blocks from 3 frames and emits new filtered blocks with low temporal noise. This allows for the creation of non-displayable, synthetic *alternate reference frames* [6, 63] that improves overall compression, and is a feature present in VP8, VP9, and AV1. The temporal filter can be iteratively applied to filter more than 3 frames, providing an additional quality/speed trade-off.

The final stage of the pipeline takes the decoded output of the encode block and applies *loop filtering* and lossless *frame buffer compression*. The former requires access to pixels from adjacent and top blocks, which are stored in local SRAM line buffers. The latter losslessly compresses each macroblock with a proprietary algorithm that minimizes memory bandwidth while staying fast enough not to be a bottleneck. The frame buffer compression reduces reference frame memory read bandwidth by approximately 50%.

Data Flow and Memory System: The *DRAM reader* block interfaces to the NoC subsystem, and is responsible for fulfilling requests for data from other blocks, primarily the *reference store*. This block also includes the *preprocessor* and *frame buffer decompression* logic. Similarly the *DRAM writer* block interfaces to the NoC subsystem for writes to DRAM.

The most memory-intensive element of video encoding, as noted earlier, is in the motion estimation stage, to find blocks of pixels from the reference frames most similar to the current block. VP9 allows blocks from multiple reference frames to be combined, further increasing the search space. Consequently, a key element of our design is an SRAM array *reference store* that holds the motion search window. A reference store of 144K[4] pixels can support each pixel (macroblock) in a tile column to be loaded exactly once during that column's processing and a maximum of twice during the frame's processing[5]. The reference store supports LRU eviction.

Given the deterministic DRAM access pattern, our design can deeply prefetch the needed macroblocks, resulting in high memory subsystem latency tolerance and maximizing memory-level parallelism. Additionally, the local search memory allows for an exhaustive, multi-resolution motion search (down to 1/8th pixel resolution), achieving higher throughput and better results than are typically obtained in a software motion estimation implementation.

The architecture of the encoding core eliminates most memory hazards, allowing for an out of order memory subsystem. In particular, all the inputs (reference buffers, input frame) are not modified during encoding, the encoded frame is written sequentially, and the decoded version of the newly encoded frame (which will become a reference frame for the next frame) is also written sequentially. The primary hazard is the use of cross-tile boundary macroblocks for the in-loop deblocking filter, which is avoided by a memory barrier at the end of each tile column. Consequently, each core in our design can have dozens of outstanding memory operations in flight. The architecture aligns accesses to the natural memory subsystem stride and does full writes to avoid read-modify-write cycles in the DRAM subsystem.

Control and Stateless Operation: The encoder IP block is programmed via a set of control/status registers for each operation. All inputs – the frame to be encoded, all reference frames, other auxiliary inputs (quantization parameter look-up tables, probability tables, temporal motion vectors) – are stored in VCU DRAM, as are all the outputs – the encoded frame, the updated reference frame, temporal motion vectors and updated probability tables. This allows the encoder cores to be interchangeable resources, where the firmware can dispatch work to any idle core. The bandwidth overhead from transferring state from DRAM is relatively small compared to the bandwidth needed to load reference frames, as discussed above. While an embedded encoder (in a camera, for example) might prefer to retain state across frames to simplify processing its single stream, this stateless architecture is better for a data center ASIC where multiple streams of differing resolutions and frame rates (and hence processing duration) are interleaved.

3.3 System Balance and Software Co-Design

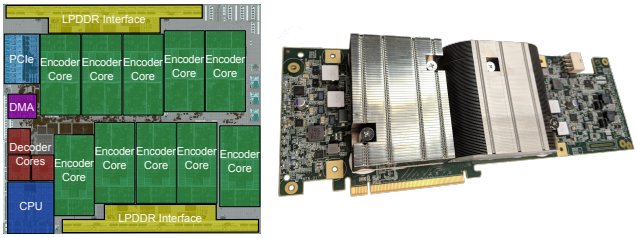
We next discuss how we brought the hardware together in an optimal *system* balance, and elaborate on the co-design across hardware and software.

3.3.1 Provisioning and System Balance: The VCU ASIC floorplan is shown in Figure 5a and comprises 10 of the encoder cores discussed in Section 3.2. All other elements are off-the-shelf IP blocks[6]. VCUs are packaged on standard full-length PCI Express cards (Figure 5b) to allow existing accelerator trays and hosts to be leveraged. Each machine has 2 accelerator trays (similar to Zhao et al. [66]), each

[4] 144K pixels = 768 pixels wide and 192 pixels tall. The width of 768 pixels represents a maximum tile column width of 512 pixels (8x84-pixel macroblocks) and a 128 horizontal search window on each side (most video motion is horizontal, so search is biased in that direction). The height of 192 pixels includes the 64-pixel macroblock and two 64-pixel windows vertically.

[5] For H.264, which lacks tile columns, the reference store is configured as a raster store of 64x16 pixel blocks. By increasing the reference store to 394K pixels (2048 x 128), the core can provide efficient encoding for up to 2048 pixel wide videos.

[6] The decoder cores are off-the-shelf, but SRAM ECC was added for data center use.



(a) Chip floorplan (b) Two chips on a PCBA

Figure 5: Pictures of the VCU

containing 5 VCU cards, and each VCU card contains 2 VCUs, giving 20 VCUs per host. Each rack has as many hosts as networking, physical space, and cluster power/cooling allow.

In terms of speeds and feeds, VCU DRAM bandwidth was our tightest constraint. Each encoder core can encode 2160p in real-time, up to 60 FPS (frames-per-second) using three reference frames. The throughput scales near-linearly with reduced pixel count from lower resolutions. At 2160p, each raw frame is 11.9 MiB, giving an average DRAM bandwidth of 3.5 GiB/s (reading one input frame and three references and writing one reference). While the access pattern causes some data to be read multiple times, the lossless reference compression reduces the worst-case bandwidth to ~3 GiB/s and typical bandwidth to 2 GiB/s. The decoder consistently uses 2.2 GiB/s, so the VCU needs ~27-37 GiB/s of DRAM bandwidth, which we provide with four 32b LPDDR4-3200 channels (~36 GiB/s of raw bandwidth). These are attached to six x32 DRAM chips, with the additional capacity used for side-band SECDED ECC [26].

Other system resources to be balanced were VCU DRAM capacity (the 8 GiB usable capacity gave modest headroom for all workloads) and network bandwidth (only 2/3 loaded in a pathological worst-case). Host CPU cores, DRAM capacity, DRAM bandwidth, and PCI Express bandwidth were also evaluated but found to be indirectly bound by network bandwidth, needing at most 1/3 of the system resources. Appendix A provides a more detailed discussion of these system balance considerations.

3.3.2 Co-Design for Fungibility and Iterative Design: The software and hardware were loosely coupled to facilitate parallel development pre-silicon and continuous iteration post-silicon. The codec cores in the VCU are programmed as opaque memories by the on-chip management firmware (the firmware and driver stack are oblivious to their content). The management firmware exposes userspace mapped queues that expose 4 commands: run-on-core, copy-from-device-to-host, copy-from-host-to-device, and wait-for-done. Notably, run-on-core does not specify a particular core, leaving it to the firmware to schedule.

We designed the system assuming that multiple userspace processes would be needed to reach peak utilization at the VCU level since we use a process-per-transcode model and the VCU is fast enough to handle multiple simultaneous streams. The firmware schedules work from queues in a round-robin way for fairness (ensuring forward progress) and to maximize utilization. Software describes the work as a data dependency graph which allows operations to start and end out-of-order while respecting dependencies between them. Typically, each userspace process controls one firmware queue with multiple threads multiplexed onto it. One thread enqueues commands to decode video in response to the need for new frames, while another enqueues commands to scale or encode video as frames become available. The loose coupling allows userspace software to adjust the flow of frames through codecs, efficiently expressing 2-pass encodes (low-latency, lagged, or offline) and changing codec modes (scaling, temporal filtering, H.264, VP9) without requiring other system changes.

It is substantially easier to iterate on userspace software in data centers than on any lower level software (firmware, kernel) because low level software updates require disruptions such as machine reboots and therefore take longer to roll out globally. Userspace VCU programming was vital for rapid iteration on the rate control algorithms after initial deployment (results in Section 4.3).

3.3.3 Co-Design for Work Scheduling & Resiliency: To realize the maximum per-VCU and data center-wide VCU utilization, we moved our video processing scheduler from a uniform CPU cost model (fixed CPU-seconds/seconds per graph step) to an *online multi-dimensional bin-packing scheduler* [19]. This ensures that no single VCU becomes completely saturated and no video transcoding task (a step in the dependency graph) becomes resource starved. Each cluster has *multiple logical "pools"* of computing defined by use case (upload, live) and priority (critical, normal, batch) that trade-off resources based on each pool's demand. Each pool has its own scheduler and multiple workers of different types (e.g. transcoding, thumbnail extraction, generating search signals, fingerprinting, notifications, etc), some with exclusive access to a VCU and some doing regular CPU based processing.

Each type of worker defines its own set of *named scalar resource dimensions* and a capacity for each. For example, resources for the VCU workers include fractional decode and encode cores, DRAM bytes, and fractional host CPU. We also use *synthetic resources* to provide an additional level of control (for example, to limit the amount of software decode to indirectly save PCI Express bandwidth which is otherwise hard to attribute to a specific process). CPU processing workers use the same scheduler but most retain the prior one-dimensional "single slot per graph step" model with the configured worker size (RAM, CPU) and per step average resource usage determining the number of available slots per worker.

The worker type also defines a *mapping* from a step request (which includes input video dimensions, input format, output formats, encoding parameters) to the amount and type of resource required. The VCU estimations were initially based on measurements of representative workloads in an unconstrained environment and then tuned using production observations. This per-worker type mapping admits different resource costs for dynamic tuning and future VCU and CPU changes. As an example, for the VCU, this mapping enabled *opportunistically boosting encoder utilization* to dynamically leverage the host CPU for decoding (based on a synthetic resource dimension discussed earlier) when hardware decoding became a resource bottleneck. (Additional agility results are discussed in Section 4.3.)

The scheduler is horizontally scaled due to the large number of workers and the need for low latency. It maintains a sharded, in-memory availability cache of all workers and their current resource capacity across all dimensions, distributing the work and

periodically receiving updates from the workers about their available resources. Work is distributed using a load-maximizing greedy scheduling algorithm across all resources. This causes workers to become idle when pool-level usage drops, at which point they may be stopped and reallocated to other pools in the cluster, maximizing cluster-wide VCU utilization. Another part of the scheduler sizes the workers based on workload mix demand.

In the event of an error, the work is rescheduled on another VCU or with software transcoding, leveraging the existing video processing framework retry mechanism.

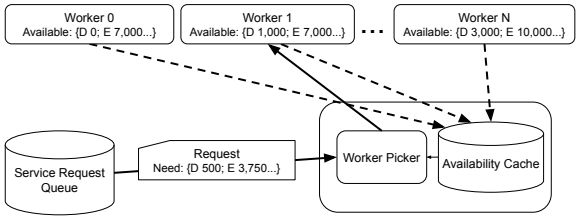


Figure 6: Video processing work scheduler

Figure 6 illustrates our design with VCU workers showing just the decoder (D) and encoder (E) core dimensions. To avoid fractions, the core dimensions use *millicores* so each VCU has 3,000 millidecode cores and 10,000 milliencode cores available. In the example, the Request requires 500 millidecode cores and 3,750 milliencode cores. The worker picker sees in its availability cache that Worker 0 has insufficient decode resources and thus will schedule the Request on Worker 1 (first fit by worker number). Worker N is fully idle and thus is a candidate for being stopped to free up resources.

3.4 High-Level Synthesis for Agility

The state-of-the-art hardware design flow adopted in the VCU development is a combination of Mentor Graphics' *Catapult* [52] tool and an in-house integration tool called Taffel that creates the C++ and Verilog fabric for composing individual HLS leaf blocks.

We implemented the encoder core design using a C++ based HLS design flow for faster development and design iteration and to avoid maintaining a separate architectural simulation model [56]. C++ development enabled the use of LLVM's AddressSanitizer [50] and MemorySanitizer [54] to discover hardware issues (out-of-bounds access, use-of-uninitialized-data, etc.) that would have been infeasible to find with conventional RTL simulation or emulation environments due to the size and duration of the tests. These issues mapped directly back to architectural errors that could be easily fixed and verified, typically in under a day.

During pre-silicon verification, our video processing system ran full quality regressions using HLS C++ simulations across our original, internal-only, large user-generated content corpus in around 1 week. Initial results showed severe quality problems; after tuning and improving the software (e.g., rate control) and re-evaluating, the quality of the hardware was sufficient to tape-out the VCU. This design methodology allowed for the first silicon tape out to be immediately usable for transcoding at scale. In addition, our design flow had several other significant benefits, discussed below.

High Productivity and Code Maintainability: With HLS, there was 5-10x less code to write, review, and maintain compared to a traditional Verilog approach. We could implement more encoder features with the available time and engineering resources.

Massively Accelerated Verification: Relying on standard software development tool flows, the C++ design exploited cloud compute for embarrassingly parallel verification. As a result, testing throughput was multiplied by 7-8 orders of magnitude over RTL simulation. HLS exposed over 99% of the functional bugs during C++ testing, before ever running full VCU RTL simulation.

Focusing Engineering Effort on High-Value Problems: Cycle-by-cycle data path control logic was designed by the HLS compiler, and we spent more time on algorithm and (macro)architecture design. We skipped the strenuous verification of the microarchitecture since the HLS flow does not suffer the human errors that ail traditional Verilog designs.

Design Space Exploration: Due to the significant microarchitectural design and verification effort with Verilog, there is often only time to evaluate one RTL design, limiting the architecture design. With our flow, we were able try numerous architectures and algorithms to find optimal quality-silicon area trade-offs for the numerous design choices in many encoding problems (motion estimation, block partitioning, and rate-distortion optimization). This led to significant gains in encoder compression efficiency while allowing us to stay within the area budget.

Late Feature Flexibility: We were able to make architectural adjustments late in the project to support late feature requests and address challenges exposed in the place and route stage of the physical design. For subsequent chip designs, our design flow will make migration to new silicon process nodes and clock frequency targets effortless.

While manual RTL may have saved some VCU silicon, the aforementioned benefits overwhelmingly tip the scales and we believe HLS was the right choice for this design (given the somewhat relaxed power and die-area constraints for data center ASICs). Furthermore, in cases where a legacy Verilog reference design was available, the HLS implementations reached silicon area parity.

3.5 Discussion

As can be seen from the prior discussions, the distinct requirements of cloud video workloads (diversity, throughput, quality) and warehouse-scale environments (efficiency, reliability, agility, scale) combined with new degrees of freedom (relaxed power/area constraints, multi-ASIC solutions, and hardware-software co-design) lead to distinct design innovations and engineering optimizations, both at the overall holistic system level and for individual components. Below, we summarize how our resulting warehouse-scale video acceleration system design is fundamentally different from consumer-centric designs in significant ways.

From a data center perspective, our VCU ASIC implements a more sophisticated encoder pipeline with more area-intensive optimizations (like temporal filtering and an aggressive memory system) and embraces density across multiple encoder and decoder cores. But at the same time, some aspects are simplified. Only the most compute-intensive aspects of the algorithm are ossified in hardware, with software fall-back (on general-purpose CPUs) for

infrequently-used or dynamically-changing computations. Similarly, resiliency mechanisms are simpler at the ASIC level (e.g., SRAM error detection in the encoder cores), relying instead on high levels of redundancy and software failure management. In addition, the VCU supports stateless operation and user-space firmware control, to provide fungibility and programmability with minimal disruption to traditional data center deployments. This can be leveraged at higher levels of the system for interoperable scheduling and continuous tuning. We also use high-level synthesis to design our ASICs for more sophisticated verification and design exploration, as well as late-feature flexibility.

We assemble multiple VCU ASICs in bigger systems and optimize the provisioning and system balance across computing, memory, and networking to match the diversity and fast-changing requirements of data center workloads. At the same time, with hardware-software co-design, we provide fungible units of work at the ASIC-level and manage these as cluster-level logical pools for novel work shapes and continuously evolving applications. Our design supports computationally-intensive multiple-output transcoding (MOT) jobs and our scheduler features rich abstractions and a new bin-packing algorithm to improve utilization.

4 DEPLOYMENT AT SCALE

Below, we evaluate our design. Section 4.1 quantifies the performance and quality improvement of our system on the public vbench benchmark, followed by fleetwide results on production workloads in Section 4.2. Sections 4.3 and 4.4 evaluate our co-design approach in post-deployment tuning and in managing failures, and Section 4.5 concludes with a discussion of new workloads and application capabilities enabled by hardware acceleration.

4.1 Benchmarking Performance & Quality

Experimental Setup: We study accelerator performance and efficiency using *vbench* [39]. This public benchmark suite consists of a set of 15 representative videos grouped across a 3-dimensional space defined by resolution, frame rate, and entropy. We load the systems under test with parallel *ffmpeg* [14] transcoding workloads processing vbench videos and we measure throughput in pixels encoded per second (Mpix/s), which allows comparison across a mix of resolutions[7].

In comparing to alternative approaches, we faced a few key challenges. Notably, our accelerator's target perceptual quality and bitrate trade-offs differed from the off-the-shelf accelerators available during the VCU's development. So, it was important to go beyond pure throughput comparisons to include quality for an accurate comparison.

We studied two baselines: a dual-socket server with Intel Skylake x86 CPUs and 384 GiB of DRAM, and a system with 4 Nvidia T4 GPUs with the dual-socket server as the host. We compare these to our production acceleration system with 10 cards (20xVCU) but also present data for an accelerator system with 4 cards given the 4-card GPU baseline. In the GPU and accelerator systems, all video transcoding is offloaded to the accelerators, and the host is only running the ffmpeg wrapper, rate control and the respective device drivers. Inherent in any comparison like this are differences in technology nodes and potential disadvantages to off-the-shelf designs from not having access to the software for co-design, etc. But, we nonetheless present comparisons with other accelerators to quantify the efficiency of the accelerator relative to state-of-the-art alternatives in that time frame.

Table 1: Offline two-pass single output (SOT) throughput in VCU vs. CPU and GPU systems

System	Throughput [Mpix/s]		Perf/TCO[8]	
	H.264	VP9	H.264	VP9
Skylake	714	154	1.0x	1.0x
4xNvidia T4	2,484	—	1.5x	—
8xVCU	5,973	6,122	4.4x	20.8x
20xVCU	14,932	15,306	7.0x	33.3x

Encoding Throughput: Table 1 shows throughput and perf/TCO (performance per total cost of ownership) for the four systems and is normalized to the perf/TCO of the CPU system. The performance is shown for offline two-pass SOT encoding for H.264 and VP9. For H.264, the GPU has 3.5x higher throughput, and the 8xVCU and 20xVCU provide 8.4x and 20.9x more throughput, respectively. For VP9, the 20xVCU system has 99.4x the throughput of the CPU baseline. The two orders of magnitude increase in performance clearly demonstrates the benefits of our VCU system.

In fact, our production workload is largely MOT, which was not supported on our GPU baseline. Prior to VCU, the production workload used multiple SOTs instead of running MOT on CPU given the high latency. MOT throughput is 1.2-1.3x higher than SOT (976 Mpix/s on H.264 and 927 Mpix/s on VP9), stemming from the single decode that is reused to produce all the output resolutions.

Given that the accelerators themselves are a non-trivial additional cost to the baseline, we use *perf/TCO* as one metric to compare the systems. We compute perf/TCO by dividing the achieved performance by the total cost of ownership (TCO)[9] which is the capital expense plus 3 years of operational expenses, primarily power. For H.264 encoding, the perf/TCO improvement of the VCU system over the baseline is 4.4x with 4 cards, and 7.0x in the denser production system. By comparison, the GPU option is a 1.5x improvement over the baseline. The cost of the GPU is driven by many features that are not used by video encoding, but at the time of development, it was the best available off-the-shelf option for offloading video encoding. For VP9 encoding, VCU improves perf/TCO over the baseline by 20.8-33.3x depending on the card density. VP9 is more computationally expensive than H.264, as can be seen in the raw throughput measurements on the baseline Skylake system, making an accelerator an even more attractive option for that format.

In a perf/watt comparison of the systems, the VCU system achieves 6.7x better perf/watt than the CPU baseline[10] for single output H.264, and 68.9x higher perf/watt on multi-output VP9.

Encoding Quality: Using the vbench microbenchmark, we compare the encoding quality of the VCU (both H.264 and VP9) versus

[7] Megapixels per second – Mpix/s – is computed by multiplying the throughput in frames per second by the width and height, in pixels, of the encode output(s).

[8] Perf/TCO is relative to the Skylake baseline with both sockets used.

[9] We are unable to discuss our detailed TCO methodology due to confidentiality reasons. At a high-level, our approach parallels TCO models discussed in prior work [4].

[10] We use only active power for the CPU system, subtracting idle. We did not collect active power for the GPU, hence we do not report those comparisons.

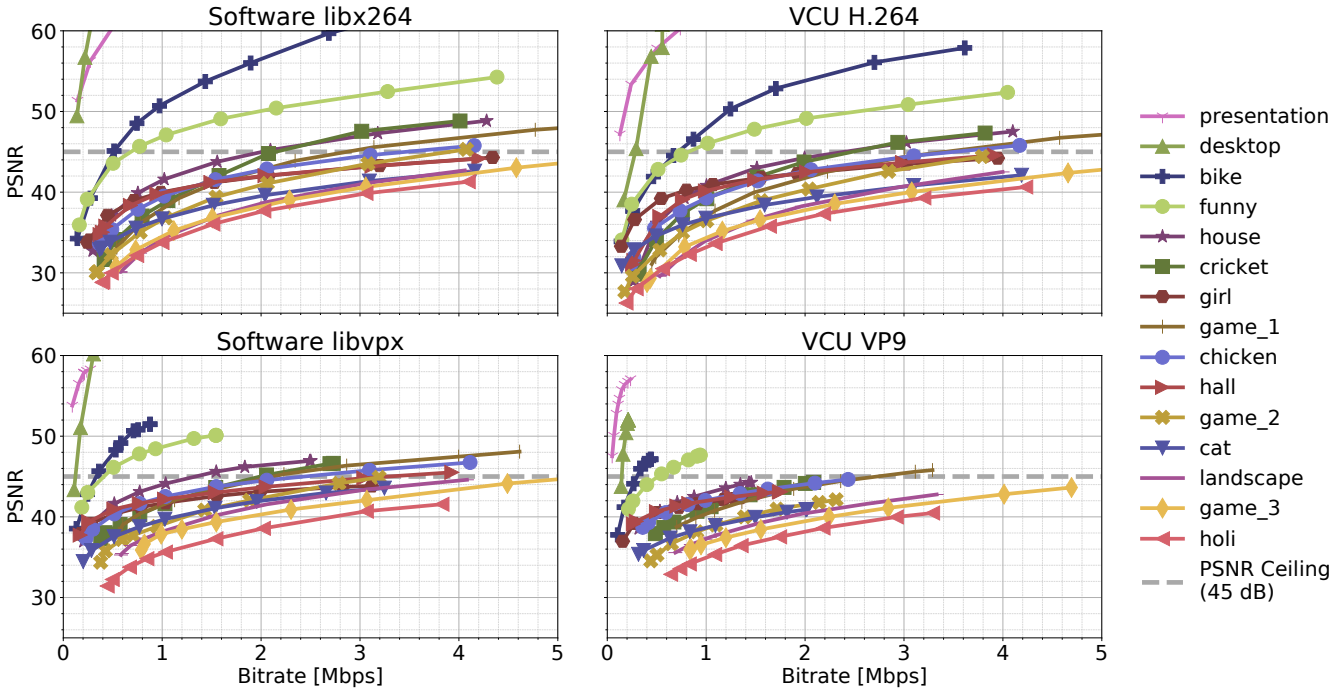


Figure 7: Rate-distortion (RD) curves comparing VCU encodings of the vbench video suite, to software libx264 and libvpx[11].

software encoding using libx264 and libvpx. Figure 7 shows the *operational rate-distortion (RD) curves* [44, p. 26–28] for each vbench video, with peak signal-to-noise ratio (PSNR) representing distortion on the vertical axis, bitrate (bits per second of video stream) on the horizontal axis. Each line is a video in the vbench suite, with points on the curves formed by encoding each video at a different target bitrate. RD curves are effective visualizations of the nature of lossy video encoding; encoders may represent a video using more or fewer bits to achieve higher or lower perceptual quality, as measured by PSNR in this case. The improvement of VP9 over H.264 is visible in Figure 7 (higher is better), where the RD curves in the bottom graphs have shifted to the left, i.e. the VP9 encoder uses fewer bits while maintaining comparable visual quality. The RD curves also show the high variance in encoding quality across videos. The topmost curves, e.g. `presentation` and `desktop`, have content that is easy to encode, i.e. high PSNR values at very low bitrates. VP9 encodings of these videos have lower PSNR values, yet this is intentional as there is minimal improvement in subjective visual quality with PSNR values above 45 dB [9]. In contrast, the bottom most curves, e.g. `holi`, are videos with a lot of motion, and they are harder to encode. For the same bitrate, the VP9 encodings of `holi` have higher PSNR than H.264.

We compare the encoding quality of VCU using BD-rate for each video relative to the software baseline, and average across the suite. BD-rate represents the average bitrate savings for the same quality (PSNR in this case) [5]. Comparing VCU-VP9 and Software-H.264 illustrates the advantages of hardware acceleration. VCU leverages the improved coding efficiency of VP9 relative to H.264 to achieve 30% BD-rate improvement relative to libx264. The high compute cost of VP9 makes it computationally infeasible at scale in software. VCU H.264 encodings are on average 11.5% higher BD-rate than libx264, and VCU VP9 is 18% greater BD-rate than libvpx. This is expected as the pipelined architecture cannot easily support all the same tools as CPU, such as Trellis quantization [49]. Section 4.3 will show how hardware bitrate has improved steadily over time via rate control tuning, such that both H.264 encoders are currently comparable.

4.2 Production Results

We next present data from fleetwide deployment and tuning, serving real production upload workloads (discussed in Section 2). Compared to the small ffmpeg benchmarks in vbench, we now measure the production transcoding service throughput, which also includes

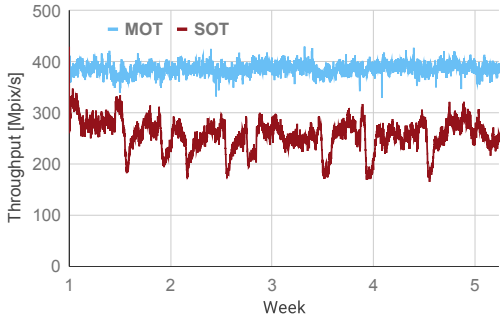
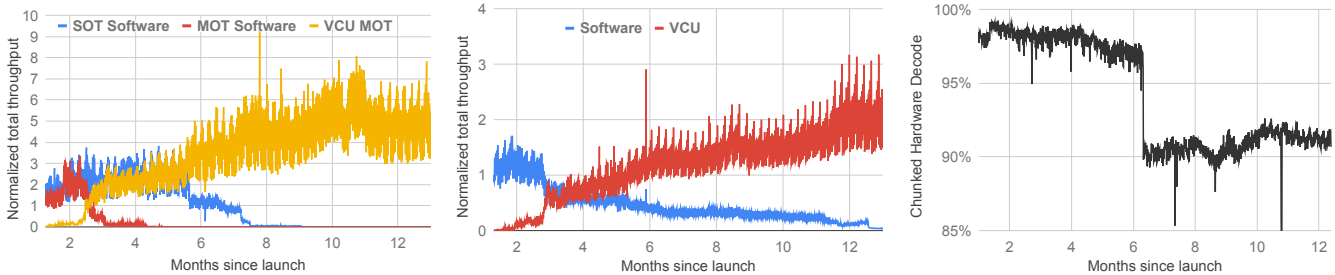


Figure 8: Throughput per VCU measured for real production video transcoding workloads

[11] The PSNR Ceiling at 45 dB depicts the limit for visually perceptible quality improvements [9].



(a) Primary upload chunked workload, 50% on VCU at launch reaching 100% in month 7

(b) Live transcoding on VCU

(c) Opportunistic software decoding to reduce hardware decode contention

Figure 9: Post-launch accelerator workload scaling.

moving video onto and off the host and a different mix of resolutions and workloads.

Figure 8 shows throughput per VCU measured in Mpix/sec for video upload workloads from production. The top, blue line is the main MOT worker job, and we observe about 400 Mpix/sec (the difference vs. vbench MOT throughput is due to I/O and workload mix). The bottom, red line is our single output transcoder and has a lower throughput of around 250 Mpix/sec because the worker must also produce inefficient low-resolution outputs for high-resolution inputs.

Performance measurements from production jobs are consistent with the trends observed with vbench: doing MOT instead of SOT is a big win. The lack of variability in the MOT line also illustrates that we are able to utilize the cores close to maximum capacity. Software encoding is no longer in production to present a concurrent fleetwide comparison, but the relative throughput of software versus hardware encoding is similar to vbench data.

4.3 Benefits of Co-Design in Deployment

Like any other hardware project, the initial measured throughput of hardware differs from predicted pre-silicon prototyping or simulation/emulation, offering substantial room for improvement post-launch. Our co-design across hardware and software created multiple opportunities for tuning. Figure 10 shows the improvement in coding efficiency on VCU for H.264 and VP9 as the percent difference in bitrate relative to software (i.e., libx264 and libvpx) from the time the accelerators were deployed. Encoder rate control runs

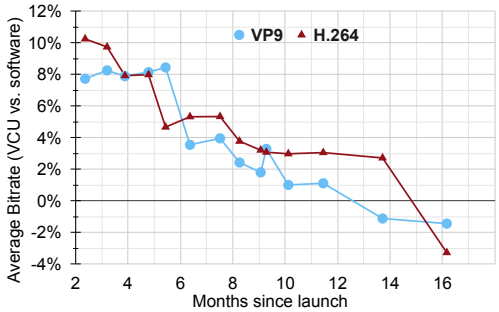


Figure 10: Hardware bitrate improvement over time relative to Software (data points weighted by per-format egress)

exclusively on the host and has improved over time, eventually surpassing software bitrates at iso-quality, including competing with improvements in software encoding. Improved group-of-pictures structure selection, better use of hardware statistics, introduction of additional reference frames, and importing rate control ideas from the equivalent software encoders were all valuable post-deployment optimizations. In many cases automated tuning tools were applied with success.

Figures 9a and 9b show the growth in total throughput per job with chunked output and live transcoding. As real workloads were deployed, various performance bottlenecks in the software stack were discovered and fixed. Continuous profiling tools at all levels of the stack (userspace, host kernel, firmware) substantially contributed to this progress. For example, measurements of loaded machines showed ~40 Gbps of average inter-socket bandwidth indicating NUMA bottlenecks, and a post-launch rollout of NUMA-aware scheduling for the accelerator jobs showed performance gains of 16-25%.

One additional benefit of the hardware-software co-design is that the scheduler can make trade-offs to reduce resource stranding. For example, Figure 9c shows how some hardware decode is shifted back to VCU host CPU to reduce encoding core stranding. We enabled this optimization after month 6 (on the horizontal axis), at which point one can see the average decoder utilization drop from approximately 98% to 91% (on the vertical axis).

4.4 Failure Management

Reliability is a first class concern [4] in the full hardware life cycle (delivery, burn-in, fault detection, repair, update, decommission). In this section, we discuss how our hardware-software co-design decisions from Section 3 helped address failure management.

Failure Monitoring and Workflow: As noted above, the VCUs are deployed in a dense configuration: 20 VCUs per host with many hosts per rack. In our warehouse environment, the rack is the unit of deployment, while the unit of repair is individual components: PCI Express cards, chassis, cables, power supplies, fans, CPUs, etc. Consequently, an individual host has dozens of discrete components that may need repair, and it is responsible for collecting fault management information from the components. The VCU firmware provides telemetry from the cards reporting various health and fault metrics (temperature, resets, ECC errors, etc). When a sufficient

number of faults have accumulated, a host will be marked as unusable and queued for repairs. To protect against faulty repair signals causing large scale capacity loss, the number of systems allowed in repair states is capped. Therefore the warehouse scheduler needs to tolerate a modest number of faulty systems in production while human technicians repair those that have been removed.

VCU Failures: It is not cost effective to send a system to repair when a small fraction of the VCUs have failed. Accordingly, the failure management system has the capability to disable an individual VCU so that the majority of the system can remain available, and the load balancing software adapts to this degradation. Given the number of encoder cores on the VCU, it would also be possible to disable individual cores, but many failure modes affect the whole VCU (e.g., DRAM errors). Managing VCUs with variable throughput due to failed cores adds to system complexity, so we chose to treat an entire VCU as the lowest level of fault management, with our board providing independent power rails for each VCU. The correlation between card swaps and VCU failures confirms that VCU failures are largely independent.

Memory/Host Failures: Given the large amount of DRAM and embedded SRAM in the transcoding system, memory corruption errors are a significant source of concern. To detect manufacturing escapes, DRAM test patterns are written and evaluated during burnin. While the DRAM has SECDED ECC, many of the embedded SRAMs only have double-error-detect, as the latency impact of error correction was unacceptable but the reliability impact was tolerable. In production, high levels of correctable or uncorrectable faults will result in disabling the VCU and will eventually trigger a repair flow (including using MABIST [3]). Failures in the host, expansion chassis, or associated cables result in the full host being disabled.

Avoiding "Black-Holing": At the system level, a failed transcoding operation on a given VCU will be retried at higher layers of software and typically assigned to a different VCU or software worker. However, a failing but not yet disabled VCU is often 'fast' relative to a working one and can naturally result in "black-holing" [35], where a disproportionate amount of traffic is sent to these bad systems. After encountering this issue in practice, we implemented the following mitigation: a transcoding worker, upon encountering a hardware failure, immediately aborts all work on the VCU, which is retried at the cluster level. A new worker, when first assigned to a VCU, does a functional reset and runs a set of short 'golden' transcoding tasks across every VCU core to detect persistent faults (relying on the core's deterministic behavior). If one is found, the worker refuses to start, preventing the bad VCU from being used until the fault management software disables the VCU or host.

Reducing "Blast Radius": As discussed earlier, videos are sharded into short chunks and are typically processed in parallel across hundreds of VCUs, so a single failing VCU can corrupt many videos. Video playback systems are generally tolerant of corruption, as broadcast media is susceptible to both erasure and corruption during transmission [29]. We still prefer to reduce this issue, so our system includes high-level integrity checks (i.e., video length must match the input) that detect and prevent most corruption. Additionally our software records the VCUs on which each chunk is processed for fault correlation. Nonetheless, the system will have bad video chunks escape, which is also seen with CPU based encoding. A future enhancement would be to use consistent hashing [32] to reduce the number of VCUs on which a given video is processed.

4.5 New Capabilities Enabled by Acceleration

Successful accelerators are not just about cost reductions but fundamentally enable new capabilities that were not previously possible. This section highlights two examples that were enabled by our VCU systems that were infeasible at scale (too expensive or too complex) with our legacy software infrastructure.

Enabling Otherwise-Infeasible VP9 Compression: As noted earlier, VP9 software encoding is typically 6-8x slower and more expensive than H.264 – a 150 frame 2160p chunk (5 seconds at 30 FPS) encoded on multiple CPU cores often takes 15 wall time minutes and over a CPU-hour. Consequently, even with chunk-level parallelism, it was infeasible from both a cost and latency perspective to produce VP9 at the time of video upload. Hence, in the non-accelerated scenario, VP9 would only be produced for the most popular videos using low-cost batch CPU after upload. Additionally, to reduce the effect of batch preemption, each resolution was produced by SOT, increasing the amount of CPU spent on re-decoding. With VCUs, we could instead shift to producing both VP9 and H.264 at upload time and leverage efficient MOT encoding.

Enabling New Use-Cases: In internet broadcasting scenarios, camera-to-eyeball delays of under 30 seconds are desirable. Our software-based encoding pipeline could produce VP9 for live streams only by encoding many short (2-second) chunks in parallel, trading end-to-end latency for throughput. As a concrete example, a 2-second 1080p chunk could be encoded in 10 seconds, the encoding system would transcode 5-6 chunks concurrently to achieve the needed throughput of a 1 video-sec/second. In practice, additional buffering was needed due to high variance in software encoding throughput. This necessarily limited the resolution, quality, and affordability of VP9; today, a single VCU can handle this MOT in real time. The consistency of the hardware transcode speed enabled an affordable 5-sec end-to-end latency stream in both H.264 and VP9. An additional new use case was Stadia, Google's cloud gaming service, which requires extremely low encoding latency at high resolution, high framerates, and excellent visual fidelity. By using the low-latency two-pass VCU based VP9 encoding, Stadia can achieve these goals and deliver 4K 60 FPS game play on connections of 35 Mbps.

5 RELATED WORK

There is a large body of work on hardware blocks for encoding/decoding (e.g., [2, 43, 60]). None of these studies discuss the block's integration into data center environments. Commercially, Ambarella [1] provides H.264 encoding (but not VP9) and Samsung Exynos [47] has support for H.264 and VP9 (but optimized for real-time transcoding, not high-quality offline two-pass). Mobile phones include encoders that are much smaller and more power-efficient, but with more relaxed quality and bitrate requirements. Our work, in addition to designing our own hardware block targeting our workload's stringent quality requirements, also takes a systems approach to designing and deploying them for the data center.

Some GPUs include support for video transcoding (e.g., one to three H.264 or H.265 encoding cores and some VP9 decoding cores), but, again, these are designed primarily for consumer applications and do not meet the quality requirements of video sharing workloads. Commodity GPU encoders provide performance and power improvements over a CPU, but the quality is only comparable to libx264 superfast up to medium settings, and notably not comparable to the high quality preset [36, 62]. Additionally, the small number of encoder cores per GPU require a very large number of host systems and cards to handle the necessary throughput posing both power and density challenges.

More broadly, there has been a large body of work on machine learning accelerators (e.g., [8, 18, 30, 38, 64]) including some that have discussed co-design across hardware and software (e.g., [23]). Similarly, there have been other studies that have examined system balance issues for warehouse-scale general-purpose workloads. The ASIC clouds paper [40] discusses assembling accelerator ASICs in large warehouse-scale environments including a simple case-study of a simple H.265 video encoder. However, their study focuses on TCO trade-offs (e.g., "two-for-two rule on ASIC NREs vs non-ASIC TCO). Accelerometer is an analytical model to evaluate acceleration opportunities in operations that are common to cloud workloads [53]. In contrast to these studies, this paper is the first work on broadcast-quality video acceleration at scale in large warehouse-scale environments, focusing at depth on the design trade-offs for commercial production workloads serving hundreds of hours of uploads per minute, as well as discussing co-design trade-offs with a production video processing software stack and deployment at scale.

Prior work for data center resource management has largely focused on the heterogeneity of the workload [13], and on the variability of performance due to interference for applications running on multicore processors [51, 58]. Scheduling for heterogeneity due to generations of servers in a data center has not covered the extreme case of accelerators [41, 46]. To the best of our knowledge, our work is the first to present data center accelerator resource management via multi-dimensional bin-packing, an approach that provides high availability, utilization, and scalability.

6 CONCLUSION

Video processing is an important and fast-growing foundational workload in warehouse-scale data centers and clouds. The exponential growth in video transcoding and storage, combined with slowing technology scaling, provide challenges around sustaining existing growth and managing costs along with opportunities to unlock new capabilities, through hardware acceleration. Video acceleration on the server side, at warehouse-scale, brings significant challenges around dealing with the workload complexity (transcoding algorithm trade-offs, quality/throughput requirements) and data-center-scale (co-design with distributed processing at scale and with high churn). In this paper, we address these challenges, presenting (to the best of our knowledge) the first work to discuss the design and deployment of video transcoding at scale in a large production fleet supporting multiple video-centric workloads (video sharing, photos/video archival, live streaming, cloud gaming) with stringent quality, throughput, latency, and cost requirements.

We present the design of our system, including a new hardware accelerator building block – the video coding unit (VCU) – and a system architecture that balances individual codec hardware blocks in VCUs, VCUs in boards and systems, all the way to individual systems in clusters and geographically-distributed data centers. We highlight how our co-design across hardware and software (at all levels from the firmware to the distributed data center scheduler) allow for improved efficiency, but, more importantly, improve fungibility and iterative design. We present results using public benchmarks and from our at-scale deployment. Our accelerator system has an order-of-magnitude performance-per-cost improvement over our prior baseline system (20x-33x) while meeting strict quality requirements, and our careful hardware-software co-design allows for real-world failure management and dynamic tuning. Our accelerator also enabled new capabilities and workloads (savings on network bandwidth and storage, live/video-on-demand workloads, cloud gaming, etc).

We believe we have only touched the tip of the iceberg on video acceleration. There are several system design trade-offs and opportunities that merit increased analysis (e.g., host computing design, accelerator disaggregation and sharing, new specifications like AV1, compiler-assisted software sanitizers applied to HLS C-simulation, etc). Similarly, rich opportunities for future innovation lie in combining transcoding with other machine-learning on video (for example, to automatically generate captions or enable video search) or, more broadly, offloading additional video processing currently applied between decoding and encoding.

ACKNOWLEDGMENTS

The systems discussed in this paper reflect the work of a many teams at Google and our partners, and we would like to acknowledge and thank all our colleagues for their contributions. We thank our our shepherd, Lizy John, and the anonymous reviewers for their valuable input in improving the paper. We would also like to thank Tom Wenisch as well as Dean Gaudet, Urs Hölzle, Mahesh Kallahalla, Pieter Kapsenberg, Martha Kim, Scott Silver, Michael Taylor, and Amin Vahdat for their input and support on earlier versions of the paper.

A SYSTEM BALANCE DETAILS

Here we present additional details on system balance that were summarized in Section 3.3.1. There is no global cluster-level performance target as each region has unique characteristics. The ideal state is based on equalizing the throughput of all clusters in a region to minimize the cost of regional redundancy while meeting demand. The lower performance bound is set by the ability to amortize the overheads that don't scale linearly with the amount of video transcoding, which includes non-transcoding resources and additional VCU racks needed for availability. Real-world constraints preclude the ideal state over extended periods of time, but these cluster-level considerations impact the target throughput of an accelerator host machine.

VCU host machines are not shared with other jobs, thus insulating workers from "noisy-neighbor" performance and availability concerns. We minimize the "data center tax" [31] by putting as many accelerators into a host as its CPU, DRAM, and network will

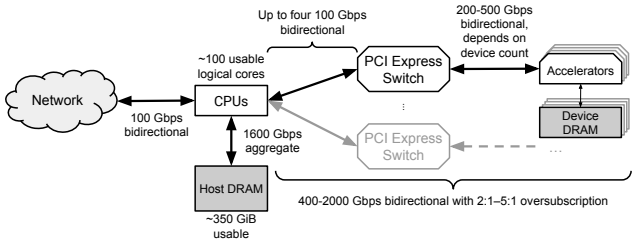


Figure 11: System balance considerations: speed & feeds

support. Combined with the throughput and cost optimized VCU design, this delivers a perf/TCO optimized accelerator system.

A.1 System-Level Design Target

To minimize development cost and risk, we reused existing components when possible. The goal was to maximize transcodes per system and minimize cost per transcode, even if certain extreme usage scenarios might incur stranding. For example, a system designed to exclusively handle 720p videos might become overloaded if it were mostly processing 144p videos (25x fewer pixels). We address these situations using the data center optimization co-design discussed in Section 3.3.2.

The host-level system constraints are shown in Figure 11. First, there is the 100 Gbps Ethernet interface that connects the host to the data center network and through which all control and video data will pass. The hosts (dual-socket Intel Skylakes) have ~100 usable logical cores, ~1600 Gbps of host DRAM bandwidth, and support up to four PCI Express expansion chassis [66] each attached with a ~100 Gbps PCI Express Gen3x16 connection. Each chassis can be configured to host between 200 and 500 Gbps of accelerator hardware. In aggregate, a system can host accelerator attachment ratios between 400 to 2000 Gbps.

A.2 Bandwidth as Transcoding Throughput

Our initial analysis of throughput suggested that the 100 Gbps network interface would be the primary constraint on the accelerator system's transcoding throughput. The worst-case output bandwidth is set by the low-latency, single-pass encoder core throughput scaled by the number of cores per host, which corresponds to a fully SOT workload (decoding is ignored).

YouTube recommends a range of upload bitrates [20], from 1 Mbps for a 360p, 30 FPS video to 68 Mbps for a 2160p, 60 FPS video, with an average of 6.1 pixels-per-bit. This gives a network interface transcoding limit of ~600 Gpixel/s per system. Allowing for double the "ideal" upload bitrates and up to 50% overheads for RPC overheads and unrelated traffic reduces this to ~153 Gpixel/s for each accelerator host, but enables tolerance of 2:1 or even 4:1 rack-level over-subscription to help reduce per-host costs based on typical bandwidth usage. The PCI Express non-video data related to operating the VCUs is <4 kiB per frame (each direction), which gives ~0.6 Gbps for 2160p and ~22 Gbps for entirely 360p video for the 153 Gpixel/s throughput, easily met by even the densest host attachment option. The encoder throughput from Section 3.3.1, equivalent to ~0.5 Gpixel/s, gives a ceiling of 30 VCUs per host for real-time or 150 VCUs for offline two-pass.

Table 2: Host resources scaled for 153 Gpixel/s throughput

Use	Logical Cores	DRAM Bandwidth
Transcoding overheads	42	214 Gbps
Network & RPC[12]	13	300 Gbps
Total	55	712 Gbps

A.3 Host CPU Usage and Memory Bandwidth

In the fully-accelerated transcoding use case, the host CPU cores are handling networking, launching new transcoding processes, muxing and demuxing the video streams, transcoding audio, and operating the accelerators. Individual transcoding processes require only a couple of MiB of system memory, as VCU DRAM holds the uncompressed video frames. Measurements for the CPU, host DRAM bandwidth, and transcoding throughput were made on existing systems (GPUs, no-op transcoding, etc.). Table 2 shows the values scaled to the above network limit, which are about half of what the target host system provides.

A.4 VCU DRAM Capacity

The primary use of DRAM during transcoding is to hold uncompressed and reference frames for both decoding and encoding. The encoder core's reference compression significantly reduces the DRAM bandwidth but slightly increases (+~5%) the DRAM footprint. The maximum expected 2160p resolution in VP9 with 10-bit color depth gives ~140 MiB for reference frames (8 plus 1 output).

As mentioned in Section 2.1, a key use-case is handling a MOT on a single VCU. The decode and encode footprint for MOT comes to ~420 MiB. Keeping up to 15 frames for lagged and offline two-pass encoding modes requires ~180-220 MiB. Padding requirements and ephemeral buffers brings the expected largest, 2160p total footprint to roughly 700 MiB per MOT and 500 MiB per SOT.

Scaling this to the network throughput limit gives a worst-case VCU DRAM requirement for low-latency SOT of 150 GiB (less than the 240 GiB of 30 VCUs) and 750 GiB for offline two-pass (less than the 1200 GiB of 150 VCUs), which supports using 8 GiB of DRAM per VCU since 4 GiB would be insufficient. The efficiency of MOT reduces these numbers by ~25% due to the reuse of decoded frames across outputs.

A.5 Aggregate System Limit

These values led us to maximize the number of VCU per expansion chassis, but concerns on the size of the failure domain and availability led us to limit each system to only two expansion chassis. Optimizing the cores per VCU led us to put two VCUs per PCI Express Gen3x16, giving 20 VCUs per host system. These conservative choices, specifically made to optimize time-to-market velocity, are well under the limits discussed above for the network, PCI Express, and host system levels. This headroom made it easier to maximize accelerator utilization, but it also left the door open to changing the system configuration after gaining production experience.

[12] 25 Gbps sustained with bursts to 100 Gbps; needing a conservative six DRAM accesses per network byte.

REFERENCES

[1] Ambarella 2015. *Ambarella H2 Product Brief.* Ambarella. Retrieved February 13, 2021 from https://www.ambarella.com/wp-content/uploads/H2-Product-Brief.pdf

[2] Ihab Amer, Wael Badawy, and Graham Jullien. 2005. A design flow for an H.264 embedded video encoder. In *2005 International Conference on Information and Communication Technology.* IEEE, 505–513. https://doi.org/10.1109/ITICT.2005.1609647

[3] Paul H. Bardell, William H. McAnney, and Jacob Savir. 1987. *Built-in Test for VLSI: Pseudorandom Techniques.* Wiley-Interscience, USA.

[4] Luiz André Barroso, Urs Hölzle, and Parthasarathy Ranganathan. 2018. *The Datacenter as a Computer* (3 ed.). Morgan & Claypool Publishers. https://doi.org/10.2200/S00874ED3V01Y201809CAC046

[5] Gisle Bjøntegaard. 2001. Calculation of Average PSNR Differences between RD-curves. In *ITU-T SG 16/Q6 (VCEG-M33).* ITU, 13th VCEG Meeting, Austin, TX, USA, 1–4.

[6] Cheng Chen, Jingning Han, and Yaowu Xu. 2020. A Non-local Mean Temporal Filter for Video Compression. In *2020 IEEE International Conference on Image Processing (ICIP).* IEEE, 1142–1146. https://doi.org/10.1109/ICIP40778.2020.9191313

[7] Chao Chen, Yao-Chung Lin, Anil Kokaram, and Steve Benting. 2017. Encoding Bitrate Optimization Using Playback Statistics for HTTP-based Adaptive Video Streaming. arXiv:1709.08763 https://arxiv.org/abs/1709.08763

[8] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. 2014. DianNao: A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine-Learning. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '14).* Association for Computing Machinery, New York, NY, USA, 269–284. https://doi.org/10.1145/2541940.2541967

[9] Yanjiao Chen, Kaishun Wu, and Qian Zhang. 2015. From QoS to QoE: A Tutorial on Video Quality Assessment. *IEEE Communications Surveys & Tutorials* 17, 2 (2015), 1126–1165. https://doi.org/10.1109/COMST.2014.2363139

[10] Cam Cullen. 2019. *Sandvine Internet Phenomena Report Q3 2019.* Sandvine. Retrieved August 19, 2020 from https://www.sandvine.com/hubfs/Sandvine_Redesign_2019/Downloads/Internet%20Phenomena/Internet%20Phenomena%20Report%20Q32019%2020190910.pdf

[11] Cam Cullen. 2020. *Sandvine Global Internet Phenomena COVID-19 Spotlight.* Sandvine. Retrieved August 20, 2020 from https://www.sandvine.com/blog/global-internet-phenomena-covid-19-spotlight-youtube-is-the-1-global-application

[12] Peter de Rivaz and Jack Haughton. 2019. *AV1 Bitstream & Decoding Process Specification.* The Alliance for Open Media. Retrieved February 13, 2021 from https://aomediacodec.github.io/av1-spec/av1-spec.pdf

[13] Christina Delimitrou and Christos Kozyrakis. 2013. Paragon: QoS-Aware Scheduling for Heterogeneous Datacenters. In *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '13).* Association for Computing Machinery, New York, NY, USA, 77–88. https://doi.org/10.1145/2451116.2451125

[14] FFmpeg developers. 2021. *FFmpeg: A complete, cross-platform solution to record, convert and stream audio and video.* FFmpeg.org. https://ffmpeg.org/

[15] John Dilley, Bruce Maggs, Jay Parikh, Harald Prokop, Ramesh Sitaraman, and Bill Weihl. 2002. Globally distributed content delivery. *IEEE Internet Computing* 6, 5 (2002), 50–58. https://doi.org/10.1109/MIC.2002.1036038

[16] Sadjad Fouladi, John Emmons, Emre Orbay, Catherine Wu, Riad S. Wahby, and Keith Winstein. 2018. Salsify: Low-Latency Network Video through Tighter Integration between a Video Codec and a Transport Protocol. In *Proceedings of the 15th USENIX Conference on Networked Systems Design and Implementation (NSDI'18).* USENIX Association, USA, 267–282.

[17] Sadjad Fouladi, Riad S. Wahby, Brennan Shacklett, Karthikeyan Vasuki Balasubramaniam, William Zeng, Rahul Bhalerao, Anirudh Sivaraman, George Porter, and Keith Winstein. 2017. Encoding, Fast and Slow: Low-Latency Video Processing Using Thousands of Tiny Threads. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17).* USENIX Association, Boston, MA, 363–376. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/fouladi

[18] Mingyu Gao, Jing Pu, Xuan Yang, Mark Horowitz, and Christos Kozyrakis. 2017. TETRIS: Scalable and Efficient Neural Network Acceleration with 3D Memory. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '17).* Association for Computing Machinery, New York, NY, USA, 751–764. https://doi.org/10.1145/3037697.3037702

[19] M.R Garey, R.L Graham, D.S Johnson, and Andrew Chi-Chih Yao. 1976. Resource constrained scheduling as generalized bin packing. *Journal of Combinatorial Theory, Series A* 21, 3 (1976), 257–298. https://doi.org/10.1016/0097-3165(76)90001-7

[20] Google, Inc. 2017. *Recommended upload encoding settings.* Google, Inc. Retrieved Feburary 13, 2021 from https://support.google.com/youtube/answer/1722171

[21] Adrian Grange, Peter de Rivaz, and Jack Haughton. 2016. *Draft VP9 Bitstream and Decoding Process Specification.* Google. Retrieved February 13, 2021 from https://www.webmproject.org/vp9/

[22] Dan Grois, Detlev Marpe, Amit Mulayoff, Benaya Itzhaky, and Ofer Hadar. 2013. Performance comparison of H.265/MPEG-HEVC, VP9, and H.264/MPEG-AVC encoders. In *2013 Picture Coding Symposium (PCS).* IEEE, 394–397. https://doi.org/10.1109/PCS.2013.6737766

[23] Kaiyuan Guo, Song Han, Song Yao, Yu Wang, Yuan Xie, and Huazhong Yang. 2017. Software-Hardware Codesign for Efficient Neural Network Acceleration. *IEEE Micro* 37, 2 (2017), 18–25. https://doi.org/10.1109/MM.2017.39

[24] Liwei Guo, Jan De Cock, and Anne Aaron. 2018. Compression Performance Comparison of x264, x265, libvpx and aomenc for On-Demand Adaptive Streaming Applications. In *2018 Picture Coding Symposium (PCS).* IEEE, 26–30. https://doi.org/10.1109/PCS.2018.8456302

[25] Lei Guo, Enhua Tan, Songqing Chen, Zhen Xiao, and Xiaodong Zhang. 2008. The Stretched Exponential Distribution of Internet Media Access Patterns. In *Proceedings of the Twenty-Seventh ACM Symposium on Principles of Distributed Computing (PODC '08).* Association for Computing Machinery, New York, NY, USA, 283–294. https://doi.org/10.1145/1400751.1400789

[26] R. W. Hamming. 1950. Error detecting and error correcting codes. *The Bell System Technical Journal* 29, 2 (1950), 147–160. https://doi.org/10.1002/j.1538-7305.1950.tb00463.x

[27] John Hennessy and David Patterson. 2018. A new golden age for computer architecture: Domain-specific hardware/software co-design, enhanced security, open instruction sets, and agile chip development. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA).* IEEE, 27–29. https://doi.org/10.1109/ISCA.2018.00011

[28] International Telecommunication Union 2019. *H.264: Advanced Video Coding for generic audiovisual services.* International Telecommunication Union. Retrieved February 13, 2021 from https://www.itu.int/rec/T-REC-H.264-201906-I/en

[29] Jae-Won Suh and Yo-Sung Ho. 2002. Error concealment techniques for digital TV. *IEEE Transactions on Broadcasting* 48, 4 (2002), 299–306. https://doi.org/10.1109/TBC.2002.806797

[30] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. 2017. In-Datacenter Performance Analysis of a Tensor Processing Unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA '17).* Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3079856.3080246

[31] Svilen Kanev, Juan Pablo Darago, Kim Hazelwood, Parthasarathy Ranganathan, Tipp Moseley, Gu-Yeon Wei, and David Brooks. 2015. Profiling a Warehouse-Scale Computer. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture (ISCA '15).* Association for Computing Machinery, New York, NY, USA, 158–169. https://doi.org/10.1145/2749469.2750392

[32] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. 1997. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing.* Association for Computing Machinery, 654–663. https://doi.org/10.1145/258533.258660

[33] Ioannis Katsavounidis. 2018. *Dynamic optimizer - a perceptual video encoding optimization framework.* Netflix. Retrieved August 19, 2020 from https://netflixtechblog.com/dynamic-optimizer-a-perceptual-video-encoding-optimization-framework-e19f1e3a277f

[34] Anil Kokaram, Thierry Foucu, and Yang Hu. 2016. *A look into YouTube's video file anatomy.* Google, Inc. https://www.googblogs.com/a-look-into-youtubes-video-file-anatomy/

[35] Ramana Rao Kompella, Jennifer Yates, Albert Greenberg, and Alex C Snoeren. 2007. Detection and localization of network black holes. In *IEEE INFOCOM 2007-26th IEEE International Conference on Computer Communications.* IEEE, 2180–2188. https://doi.org/10.1109/INFCOM.2007.252

[36] Jan Kufa and Tomas Kratochvil. 2017. Software and hardware HEVC encoding. In *2017 International Conference on Systems, Signals and Image Processing (IWSSIP).* IEEE, 1–5. https://doi.org/10.1109/IWSSIP.2017.7965585

[37] Kevin Lee and Vijay Rao. 2019. *Accelerating Facebook's infrastructure with application-specific hardware.* Facebook. Retrieved August 20, 2020 from https://engineering.fb.com/data-center-engineering/accelerating-infrastructure/

[38] Daofu Liu, Tianshi Chen, Shaoli Liu, Jinhong Zhou, Shengyuan Zhou, Olivier Teman, Xiaobing Feng, Xuehai Zhou, and Yunji Chen. 2015. PuDianNao: A

Polyvalent Machine Learning Accelerator. *SIGPLAN Not.* 50, 4 (March 2015), 369–381. https://doi.org/10.1145/2775054.2694358

[39] Andrea Lottarini, Alex Ramirez, Joel Coburn, Martha A. Kim, Parthasarathy Ranganathan, Daniel Stodolsky, and Mark Wachsler. 2018. vbench: Benchmarking Video Transcoding in the Cloud. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '18)*. Association for Computing Machinery, New York, NY, USA, 797–809. https://doi.org/10.1145/3173162.3173207

[40] Ikuo Magaki, Moein Khazraee, Luis Vega Gutierrez, and Michael Bedford Taylor. 2016. ASIC Clouds: Specializing the Datacenter. In *Proceedings of the 43rd International Symposium on Computer Architecture (ISCA '16)*. IEEE Press, 178–190. https://doi.org/10.1109/ISCA.2016.25

[41] Jason Mars and Lingjia Tang. 2013. Whare-Map: Heterogeneity in "Homogeneous" Warehouse-Scale Computers. In *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA '13)*. Association for Computing Machinery, New York, NY, USA, 619–630. https://doi.org/10.1145/2485922.2485975

[42] Debargha Mukherjee, Jim Bankoski, Adrian Grange, Jingning Han, John Koleszar, Paul Wilkins, Yaowu Xu, and Ronald Bultje. 2013. The latest open-source video codec VP9 - An overview and preliminary results. In *2013 Picture Coding Symposium (PCS)*. IEEE, 390–393. https://doi.org/10.1109/PCS.2013.6737765

[43] Ngoc-Mai Nguyen, Edith Beigne, Suzanne Lesecq, Duy-Hieu Bui, Nam-Khanh Dang, and Xuan-Tu Tran. 2014. H.264/AVC hardware encoders and low-power features. In *2014 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*. IEEE, 77–80. https://doi.org/10.1109/APCCAS.2014.7032723

[44] Antonio Ortega and Kannan Ramchandran. 1998. Rate-distortion methods for image and video compression. *IEEE Signal Processing Magazine* 15, 6 (1998), 23–50. https://doi.org/10.1109/79.733495

[45] Grzegorz Pastuszak. 2016. High-speed architecture of the CABAC probability modeling for H.265/HEVC encoders. In *2016 International Conference on Signals and Electronic Systems (ICSES)*. IEEE, 143–146. https://doi.org/10.1109/ICSES.2016.7593839

[46] Francisco Romero and Christina Delimitrou. 2018. Mage: Online and Interference-Aware Scheduling for Multi-Scale Heterogeneous Systems. In *Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques (PACT18)*. Association for Computing Machinery, Article 19, 13 pages. https://doi.org/10.1145/3243176.3243183

[47] Samsung 2018. *Exynos 8895 Processor: Specs, Features.* Samsung. Retrieved February 13, 2021 from https://www.samsung.com/semiconductor/minisite/exynos/products/mobileprocessor/exynos-9-series-8895/

[48] Y. Sani, A. Mauthe, and C. Edwards. 2017. Adaptive Bitrate Selection: A Survey. *IEEE Communications Surveys Tutorials* 19, 4 (2017), 2985–3014. https://doi.org/10.1109/COMST.2017.2725241

[49] H. Schwarz, T. Nguyen, D. Marpe, and T. Wiegand. 2019. Hybrid Video Coding with Trellis-Coded Quantization. In *2019 Data Compression Conference (DCC)*. IEEE, 182–191. https://doi.org/10.1109/DCC.2019.00026

[50] Konstantin Serebryany, Derek Bruening, Alexander Potapenko, and Dmitry Vyukov. 2012. AddressSanitizer: A Fast Address Sanity Checker. In *Proceedings of the 2012 USENIX Conference on Annual Technical Conference (USENIX ATC'12)*. USENIX Association, USA, 28.

[51] Daniel Shelepov, Juan Carlos Saez Alcaide, Stacey Jeffery, Alexandra Fedorova, Nestor Perez, Zhi Feng Huang, Sergey Blagodurov, and Viren Kumar. 2009. HASS: A Scheduler for Heterogeneous Multicore Systems. *SIGOPS Oper. Syst. Rev.* 43, 2 (April 2009), 66–75. https://doi.org/10.1145/1531793.1531804

[52] Siemens Digital Industries Software 2021. *Catapult High-Level Synthesis.* Siemens Digital Industries Software. Retrieved Feburary 13, 2021 from https://www.mentor.com/hls-lp/catapult-high-level-synthesis

[53] Akshitha Sriraman and Abhishek Dhanotia. 2020. Accelerometer: Understanding Acceleration Opportunities for Data Center Overheads at Hyperscale. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '20)*. Association for Computing Machinery, New York, NY, USA, 733–750. https://doi.org/10.1145/3373376.3378450

[54] Evgeniy Stepanov and Konstantin Serebryany. 2015. MemorySanitizer: Fast Detector of Uninitialized Memory Use in C++. In *Proceedings of the 13th Annual IEEE/ACM International Symposium on Code Generation and Optimization (CGO '15)*. IEEE Computer Society, USA, 46–55. https://doi.org/10.1109/CGO.2015.7054186

[55] Gary J. Sullivan and Thomas Wiegand. 2005. Video Compression - From Concepts to the H.264/AVC Standard. *Proc. IEEE* 93, 1 (2005), 18–31. https://doi.org/10.1109/JPROC.2004.839617

[56] A. Takach. 2016. High-Level Synthesis: Status, Trends, and Future Directions. *IEEE Design & Test* 33, 3 (2016), 116–124. https://doi.org/10.1109/MDAT.2016.2544850

[57] Tung-Chien Chen, Chung-Jr Lian, and Liang-Gee Chen. 2006. Hardware architecture design of an H.264/AVC video codec. In *Asia and South Pacific Conference on Design Automation, 2006.* IEEE, 8 pp.–. https://doi.org/10.1109/ASPDAC.2006.1594776

[58] K. Van Craeynest, A. Jaleel, L. Eeckhout, P. Narvaez, and J. Emer. 2012. Scheduling heterogeneous multi-cores through performance impact estimation (PIE). In *2012 39th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 213–224. https://doi.org/10.1109/ISCA.2012.6237019

[59] Abhishek Verma, Luis Pedrosa, Madhukar R. Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. 2015. Large-scale cluster management at Google with Borg. In *Proceedings of the European Conference on Computer Systems (EuroSys)*. Association for Computing Machinery, Bordeaux, France, Article 18, 17 pages. https://doi.org/10.1145/2741948.2741964

[60] K. Wei, S. Zhang, H. Jia, D. Xie, and W. Gao. 2012. A flexible and high-performance hardware video encoder architecture. In *2012 Picture Coding Symposium*. IEEE, 373–376. https://doi.org/10.1109/PCS.2012.6213368

[61] P. H. Westerink, R. Rajagopalan, and C. A. Gonzales. 1999. Two-pass MPEG-2 variable-bit-rate encoding. *IBM Journal of Research and Development* 43, 4 (1999), 471–488. https://doi.org/10.1147/rd.434.0471

[62] M. A. Wilhelmsen, H. K. Stensland, V. R. Gaddam, A. Mortensen, R. Langseth, C. Griwodz, and P. Halvorsen. 2014. Using a Commodity Hardware Video Encoder for Interactive Video Streaming. In *2014 IEEE International Symposium on Multimedia*. IEEE, 251–254. https://doi.org/10.1109/ISM.2014.58

[63] Yaowu Xu. 2010. *Inside WebM Technology: The VP8 Alternate Reference Frame.* Google, Inc. Retrieved Feburary 13, 2021 from http://blog.webmproject.org/2010/05/inside-webm-technology-vp8-alternate.html

[64] Xuan Yang, Mingyu Gao, Qiaoyi Liu, Jeff Setter, Jing Pu, Ankita Nayak, Steven Bell, Kaidi Cao, Heonjae Ha, Priyanka Raina, Christos Kozyrakis, and Mark Horowitz. 2020. Interstellar: Using Halide's Scheduling Language to Analyze DNN Accelerators. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '20)*. Association for Computing Machinery, New York, NY, USA, 369–383. https://doi.org/10.1145/3373376.3378514

[65] Yu-Wen Huang, Bing-Yu Hsieh, Tung-Chien Chen, and Liang-Gee Chen. 2005. Analysis, fast algorithm, and VLSI architecture design for H.264/AVC intra frame coder. *IEEE Transactions on Circuits and Systems for Video Technology* 15, 3 (2005), 378–401. https://doi.org/10.1109/TCSVT.2004.842620

[66] Whitney Zhao, Tiffany Jin, Cheng Chen, Siamak Taveallaei, and Zhenghui Wu. 2019. *OCP Accelerator Module Design Specification.* Open Compute Project. Retrieved February 13, 2021 from https://www.opencompute.org/documents/ocp-accelerator-module-design-specification-v1p0-3-pdf