Display Stream Compression Decoders for Fine-Grained Many-Core Processor Arrays

Shifu Wu, *Student Member, IEEE*, and Bevan M. Baas, *Senior Member, IEEE*

***Abstract*—This brief presents two software Display Stream Compression (DSC) video decoder designs for many-core processor arrays. The first design exploits fine-grained task-level parallelism and is able to decode pictures configured into one column of slices; it is implemented with 88 processors and 2 shared memory modules. The second design facilitates higher performance by leveraging scalable slice-level parallelism and is tailored for pictures configured into multiple columns of slices; one implementation of this design is mapped to 359 processors and 6 shared memory modules. At 1.75 GHz and 1.1 V, the proposed decoders decode 1080p video sequences in 4:2:0, 4:2:2, and 4:4:4 pixel formats—achieving up to 94.7 frames per second (fps), 95.6 fps, and 47.9 fps, while dissipating 23.9 nJ, 26.7 nJ, and 47.2 nJ per pixel, respectively. Our designs achieve up to 159× higher throughput and 841× lower energy per pixel than a DSC decoder implemented on one core of an Intel i7-7700HQ processor.**

***Index Terms*—Display stream compression (DSC), many-core, real-time, software, video decoder, visually lossless.**

I. Introduction

With the increasing use of high screen resolutions, high frame rates, and greater dynamic range in video applications, transmitting uncompressed pixel data over display links requires significant data traffic. For example, 120 Gbps is needed for 8K ultra-high-definition (UHD) videos with 10 bits per component (bpc) at 120 frames per second (fps). However, the bandwidth of the physical layer is not keeping pace. To address the disparity, a widely accepted solution is to reduce the required data rate by transmitting compressed video data.

The Display Stream Compression (DSC) standard [1], [2], which was developed by Video Electronics Standards Association (VESA), offers low-cost, low-latency, and visually lossless [3] video compression over display links. DSC performs intra-picture coding at a programmable bit rate of 8 bits per pixel (bpp) or higher, resulting in up to 3× compression for pictures of 8 bpc. It requires only one picture line storage and a small rate buffer; no off-chip memory is needed.

Although H.264/AVC [4] and High Efficiency Video Coding (HEVC) [5] can achieve higher coding efficiency [6] than DSC, their computation complexity [7], [8], implementation costs, and latency are higher. In addition, DSC supports more color bit depths, including 8, 10, 12, 14, and 16 bpc. Moreover, H.264/AVC Intra-only cannot achieve visually lossless quality for all types of content at 8 bpp with low hardware complexity for real-time high-throughput implementations [2]. The HEVC screen content coding extension (HEVC-SCC) [9] enhances screen coding capabilities of HEVC but it has high complexity. Comparison of DSC and HEVC-SCC is published in [10].

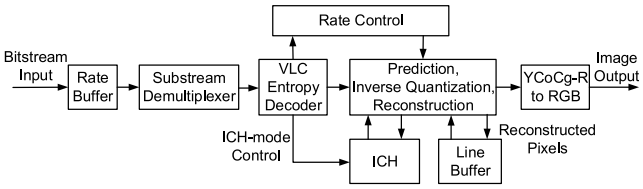

Fig. 1. The DSC decoding process, modified from [1, Figs. 3–5].

Application-specific integrated circuit (ASIC) video decoders achieve the best performance and energy efficiency, but are neither flexible nor scalable, whereas software decoders allow for full flexibility and scalability. Software video decoders implemented on single-core or multicore processors mostly utilize coarse-grained parallelism, such as at the thread level, whereas many-core computation platforms enable fine-grained task-level parallelism that leads to significant improvements over coarse-grained parallelism on performance and energy efficiency. There has been significant research on software design of H.264/AVC and HEVC decoders [8], [11], [12]. In terms of DSC, multiple hardware codecs have been published [13]–[15]; however, no software DSC decoder designs have yet been reported.

We present two software DSC decoders for programmable many-core processor arrays. By exploiting fine-grained task-level parallelism within the DSC decoding algorithm, the decoder is partitioned into small tasks, each of which is mapped to one small processor. Moreover, slice-level parallelism is applied to achieve higher decoding performance.

The remainder of this brief is organized as follows. Section II overviews the DSC decoding process and the targeted many-core processor arrays. Section III presents a slice decoder design. Section IV discusses a parallel slice decoder. Section V presents and analyzes the results. Finally, Section VI concludes the brief.

II. Background

A. Display Stream Compression (DSC) Decoding Process

DSC supports pictures in RGB, YCbCr 4:4:4, YCbCr 4:2:2, and YCbCr 4:2:0 formats. In 4:2:0 and 4:2:2 formats, every two consecutive pixels are packed as a *container pixel*, resulting in approximately twice the throughput. A pixel contains four *components* in 4:2:2 format and three components in other formats. Three adjacent pixels are defined as a *group*. DSC

Manuscript received February 6, 2021; accepted March 8, 2021. Date of publication March 23, 2021; date of current version April 30, 2021. This brief was recommended by Associate Editor J. R. Cavallaro. *(Corresponding author: Shifu Wu.)*

The authors are with the Department of Electrical and Computer Engineering, University of California, Davis, CA 95616 USA (e-mail: ucdwu@ucdavis.edu; bbaas@ucdavis.edu).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCSII.2021.3068272.

Digital Object Identifier 10.1109/TCSII.2021.3068272

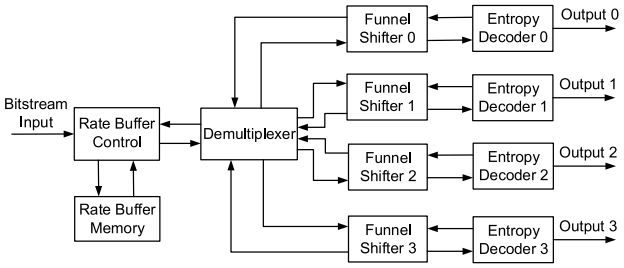

Fig. 2. Dataflow diagram of the rate buffer, substream demultiplexer, and entropy decoder.

defines *slices* as identical-sized rectangular regions which are processed independently.

DSC decoders decompress a compliant bitstream into pixels which are output in raster-scan order. Fig. 1 illustrates the decoding process. The bitstream enters the *rate buffer* (*RB*), which packs bits into fixed-length packets, called *mux words*. The *substream demultiplexer* requests mux words from the rate buffer and splits them into three or four substreams, which are parsed by the *variable length coding* (*VLC*) *entropy decoder*. The *rate control* (*RC*) module manages rate buffer fullness and calculates a *quantization parameter* (*Qp*) for every group. The *prediction, inverse quantization, and reconstruction* module inverse quantizes the quantized residuals, which are decoded by the entropy decoder, and predicts the pixel values with appropriate predictors. The inverse quantized residuals are added to the predicted values to form reconstructed pixel values. *Indexed color history* (*ICH*) keeps a record of 32 recently reconstructed pixel values, each of which is addressed by a 5-bit index. If ICH is used to code a group, each pixel is decoded as the ICH pixel pointed to by the selected index. Otherwise, the reconstructed pixels of the current group are used. The decoded pixels are written to the *line buffer* (*LB*) and read out in the next line. The *YCoCg-R to RGB* conversion occurs if RGB format output is desired.

B. The Targeted Many-Core Processor Arrays

This brief targets fine-grained multiple instruction multiple data (MIMD) many-core arrays of independent, programmable processors. The KiloCore chip [16] is an example of such a processor array. It consists of 1,000 RISC-style processors and 12 64-KB shared memory modules connected via a 2-D mesh network that supports communication between adjacent and distant processors. Each processor contains 128×40-bit words of instruction memory, 256×16-bit words of data memory, two 32×16-bit input FIFOs, and occupies 0.055 mm^2 in 32 nm CMOS technology.

III. Slice Decoder

This section presents a many-core software DSC decoder design, called *slice decoder*, which is capable of decoding pictures configured into one column of slices. Task-level parallelism is exploited to partition the decoder into small tasks. Furthermore, different pixel components are processed in parallel. Multiple data dependencies, where the results of a group are used in the following group, form feedback loops and limit performance. The latency of the loops is minimized in two steps. First, since inter-processor communication incurs latency overhead, the number of processors involved in the feedback loops is minimized by fitting as much work as possible in each processor, which in turn reduces the chip area used. Second, instructions are scheduled such that the execution time of the instructions in the loop is minimized. Area in non-critical paths is further reduced by fitting multiple tasks into one processor. For example, the YCoCg-R to RGB pixel format conversion is merged with one processor of the ICH module. Fig. 3 illustrates an example of mapping the slice decoder to 88 processors and 2 shared memory modules using an unpublished internally-developed automatic mapping tool. The tool maps the partitioned tasks to processors including considerations such as inter-processor communication patterns to reduce routing congestion and energy usage.

A. Rate Buffer and Substream Demultiplexer

Fig. 2 depicts the dataflow of the rate buffer, substream demultiplexer, and entropy decoder. Four parallel *funnel shifters* buffer the substreams. In every group, the demultiplexer receives requests from each funnel shifter indicating a mux word is needed. Then it requests mux words from the rate buffer and sends them to the funnel shifters. The time that funnel shifters spend waiting for mux words is minimized by reading mux words from the rate buffer beforehand. Thus, mux words can be sent back to funnel shifters right after the demultiplexer receives requests. The funnel shifter is updated with the newly received mux word. Then, it sends bits to the entropy decoder, which returns the actual number of bits used. The funnel shifter is updated again by removing the decoded bits. The latency of this path is optimized by sending bits to the entropy decoder right after enough bits have been updated. While waiting for data from the entropy decoder, the funnel shifter keeps updating the remaining bits in its buffer. As such, some latency in this path is hidden.

B. Entropy Decoder

As Fig. 2 shows, substreams are decoded in parallel with different entropy decoders. An entropy decoder consists of three serial tasks: residual size prediction; prefix decode; ICH index decode in ICH-mode, or quantized residual decode and size calculation in predictive mode. In addition, flatness and coding mode are decoded in the first component. The residual size prediction is based on decoded residual sizes of the previous group, which means the decoding of current group cannot start until the previous group is finished. Therefore, minimizing the total latency improves performance. We achieve this by mapping the entropy decoder into one processor per component, with the exception of the first component, where a processor is dedicated for flatness decoding.

C. Rate Control

Rate control dynamically selects a Qp for entropy decoding and prediction of the next group. It consists of five tasks: buffer level tracking, linear transformation, long-term parameter selection, short-term Qp adjustment, and flatness Qp overrides. We further partition this module into nine small tasks, each of which is mapped to one processor. Note that the entropy decoder uses Qp from the rate control to calculate input data for rate control of the next group. This loop is optimized by mapping short-term Qp adjustment and flatness Qp overrides to two and one processors, respectively.

D. Prediction, Inverse Quantization, and Reconstruction

Every group is predicted with one of the three predictors: modified median-adaptive prediction (MMAP), block

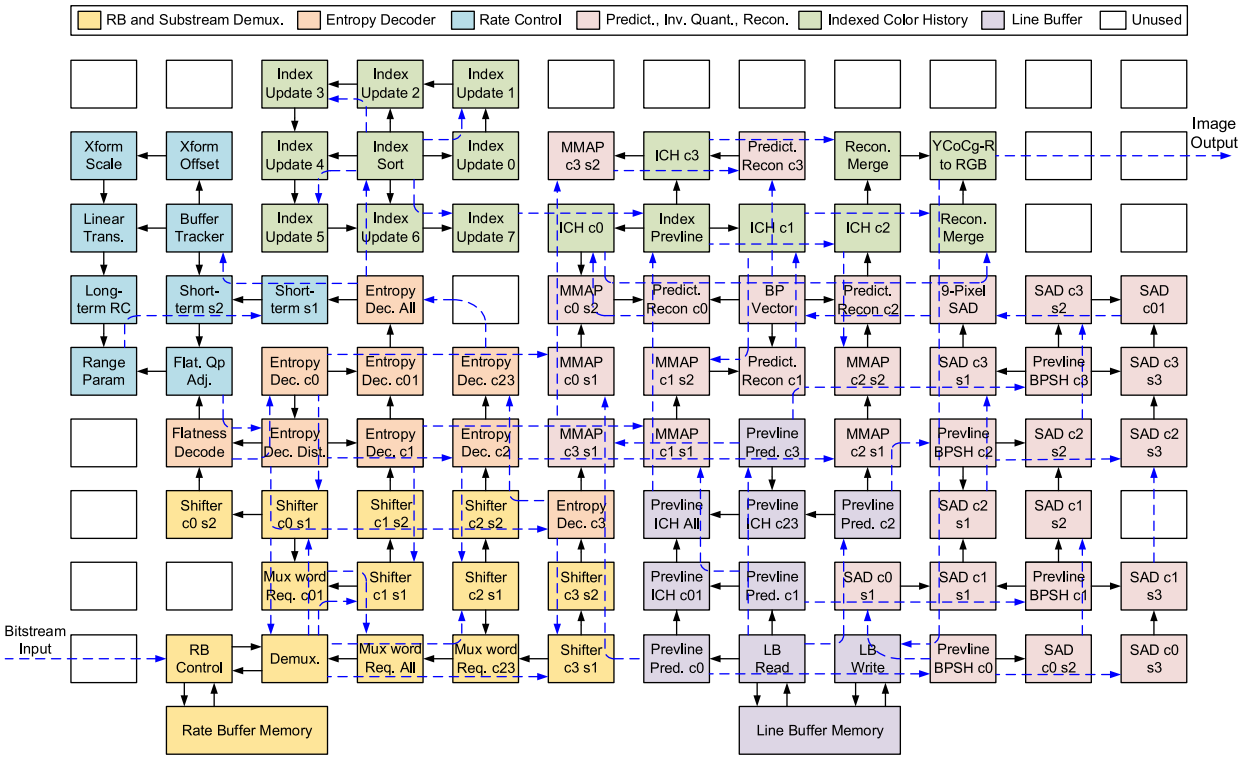

Fig. 3. Processor array mapping of the slice decoder by a mapping tool. For clarity, unused inter-processor communication links are omitted. Processors corresponding to pixel components contain "c0", "c1", "c2", or "c3" in their names; "c01" and "c23" denote processors performing merging operations. Tasks mapped to multiple processors are named with a suffix of "s1", "s2", or "s3" to denote the stage of computation.

prediction (BP), and midpoint prediction (MPP), all of which require reconstructed pixels of the previous group. Three processors are used to process one component—the first processor performs inverse quantization and the operations of MMAP which do not use reconstructed values of the previous group, and the other two processors conduct the remaining prediction operations and reconstruction. The decision between MMAP and BP is made by the *BP search* process, which calculates nine 9-pixel sum of absolute differences (SAD) using previous line reconstructed pixels. To reduce computation complexity, 3-pixel SADs are calculated for every group. Then, the 3-pixel SADs of the two previous groups are reused and added to that of the current group to form 9-pixel SADs. The optimal partition of the 3-pixel SAD calculation task, which achieves the smallest area without becoming the bottleneck in the decoder, is to use three processors to calculate the 3-pixel SAD of one component, resulting in a total of 12 processors.

E. Indexed Color History (ICH)

The main challenge of the ICH module is updating the ICH entries in every group. A straightforward approach is to manage the ICH entries as shift registers and serially shift the entries. However, even with the shifting operations performed in parallel across components, the design is still too slow. To achieve higher throughput, the ICH indices and entries are updated separately, as shown in Fig. 4. First, eight processors are used to update the indices in parallel, each of which maintains the indices of four ICH entries. Then, the ICH entries are updated by writing zero to three new pixels into the appropriate locations, and thereby the shifting operation is avoided. In the proposed slice decoder, using eight processors to update indices gives sufficient throughput. More details on parallelizing the ICH have been published [17].

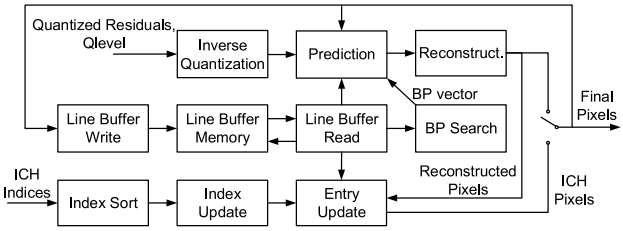

Fig. 4. Dataflow diagram of prediction, indexed color history, and line buffer.

F. Line Buffer

In this design, a shared memory is used for line buffer storage. As Fig. 4 shows, buffer write and read are partitioned into separate tasks and mapped to different processors, so that write and read can occur in parallel. The read data is distributed to four processors, each of which stores and sends the data of one component to BP search, prediction, and ICH. Constructing the previous line ICH pixels is mapped to three processors. In total, nine processors are used for the line buffer.

IV. Parallel Slice Decoder

Since the DSC standard is designed to work in raster-scan order and slices are independently decoded, pictures configured into multiple columns of slices allow for parallel decoding of slices. We propose a scalable *parallel slice decoder* design, which utilizes slice-level parallelism and decodes every column of slices with a separate *modified slice decoder*—the same as the slice decoder except without a rate buffer. Fig. 5 shows the dataflow of the parallel slice decoder. Throughput scales linearly with the number of modified slice decoders, whereas energy efficiency remains almost the same.

In the parallel slice decoder, it is essential to make sure all modified slice decoders are synchronized. In every group, the

TABLE I
COMPARISON OF SOFTWARE VIDEO DECODER IMPLEMENTATIONS*

Design	Standard	Platform	Tech. (nm)	Freq. (GHz)	Area (mm^2)	Throughput (Mpixels/s)			Throughput / Area (Mpps/mm^2)			Energy / Pixel (nJ/pixel)		
ASICON'13 [11]	H.264	24-Core Proc.	65	0.80	7.29	76.95			10.56			6.34		
TCSVT'12 [8]	HEVC	i7-3720QM	22	2.60	NA	58.38			NA			503.4†		
TCSVT'16 [12]	HEVC	i7 E5-1650	32	3.40	NA	329.7			NA			197.2†		
ICIP'20 [18]	VVC	i9-9980HK	14	2.40	NA	215.8			NA			201.1†		
VESA C Model [1]	DSC	i7-7700HQ	14	2.81	26.94‡	1.67	1.25	0.83	0.06	0.05	0.03	17,412	22,230	34,273§
Slice Decoder	DSC	KiloCore	32	1.75	5.17	49.21	49.68	24.90	**9.52**	**9.62**	**4.82**	**23.6**	**26.4**	**46.6**
Parallel Slice Decoder	DSC	KiloCore	32	1.75	20.73	**196.27**	**198.27**	**99.30**	9.47	9.57	4.79	23.9	26.7	47.2

* The area, throughput, and energy data are scaled to 32 nm using data from Holt [19]; throughput and energy data of 4:2:0, 4:2:2, and 4:4:4 pixel formats are reported in the left, middle, and right of the columns, respectively.
† Energy data are unavailable; assuming device power is half of thermal design power (TDP) [20].
‡ The core area is estimated using a publicly available die photo.
§ Energy data are estimated using Intel Power Gadget 3.5.

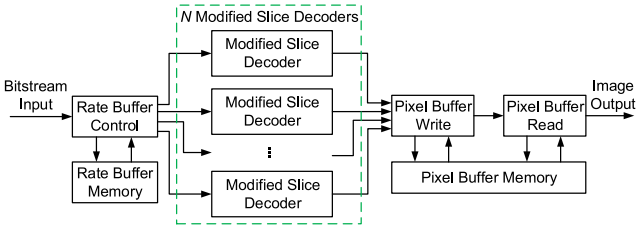

Fig. 5. Dataflow diagram of the parallel slice decoder.

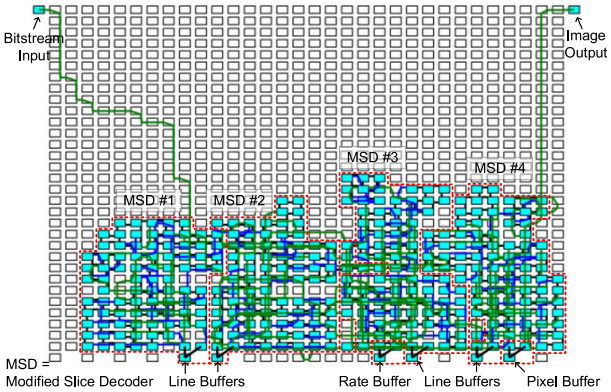

Fig. 6. Processor array mapping of the parallel slice decoder. Four line buffers, a rate buffer, and a pixel buffer are mapped to six memory modules at the bottom of the array. Input and output ports are at the top-left and top-right corners of the array, respectively. Black, blue, and green lines represent inter-processor communication links, according to the link length.

decoder reads the budgeted amount of data from the input bitstream and writes them into rate buffer memory, where every column of slices is allocated a dedicated portion of memory locations. On the buffer read side, sending a constant amount of bits to each slice decoder can cause synchronization problems, since the number of bits to code each group can vary significantly. Therefore, only the requested number of mux words are read out and sent to the modified slice decoders.

Another challenge is to rasterize the decoded pixels. Since the modified slice decoders are synchronized, and the output pixels of each slice are in raster-scan order within the slice, our solution is to implement a pixel buffer: write the output pixels of all modified slice decoders into a shared memory and read them out in raster-scan order whenever one picture line of pixels has been written. The purpose of the pixel buffer is to store the pixels of the modified slice decoders when they are not sent to decoder output, so that processors do not stall on output writes. To facilitate concurrent buffer writes and reads, separate processors are used for write and read control.

A parallel slice decoder that contains four modified slice decoders is implemented on the KiloCore processor array utilizing 359 processors (each modified slice decoder utilizes 87 processors; the control of rate buffer and pixel buffer are mapped to 4 processors; 7 additional processors are used for routing and merging data) and 6 shared memory modules. Fig. 6 shows the mapping of this design by the mapping tool. Note that the KiloCore has 1,000 processors and the decoder is mapped to the bottom half of the array.

V. RESULTS AND ANALYSIS

The slice decoder and parallel slice decoder comply with DSC v1.2a, and support 8 and 10 bpc in constant bit rate (CBR) mode. They are evaluated on a cycle-accurate C++ simulator of the KiloCore chip [21]. Both designs are configured to run at 8 bpc and 8 bpp in CBR mode, and are simulated at 1.75 GHz with a supply voltage of 1.1 V. Results from decoding three 1080p (1920×1080) pictures of different scenes are reported in Fig. 7 and Table I.

Fig. 7 shows the area and energy dissipation breakdown of the slice decoder. The area of processors and shared memory modules used in the decoder is considered in this analysis. The prediction module is the largest and accounts for one third of the total area in the decoder, due to the high computation requirement in BP search and component-level parallelism. The total area of prediction and ICH is over 50% of the entire decoder. 4:2:2 format is used for energy dissipation analysis, since the fourth component is used only in this format. Energy dissipation is correlated with the chip area used; therefore, energy breakdown is similar to the area breakdown.

Table I shows the throughput and energy results of the proposed DSC decoders. Compared to 4:4:4 format, the decoders achieve approximately 2× throughput and 50% energy per pixel in 4:2:0 and 4:2:2 formats, since two pixels are packed together and processed as one. Note that 4:2:2 format dissipates 12% more energy per pixel than 4:2:0 format, since 4:2:2 format has one more component. Dividing throughput by the number of pixels per frame results in frame rate. In 4:2:0, 4:2:2, and 4:4:4 formats of 1080p, the slice decoder achieves 23.7 frames per second (fps), 24.0 fps, and 12.0 fps,

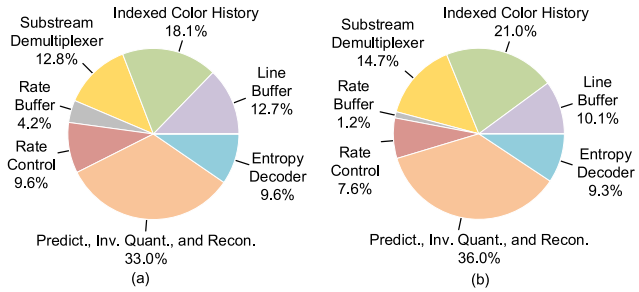

Fig. 7. Slice decoder breakdown. (a) Area. (b) Energy dissipation.

whereas the parallel slice decoder delivers 94.7 fps, 95.6 fps, and 47.9 fps, respectively. Both decoders achieve almost the same energy per pixel and throughput per chip area.

Compared to the DSC C model provided by VESA that runs on one core of an Intel i7-7700HQ processor, our designs achieve up to 159× higher throughput, 841× lower energy per pixel, and 192× higher throughput per chip area—this shows the performance and energy efficiency benefits of our many-core design approach over general purpose processors. The H.264/AVC intra-frame decoder by Zhu *et al.* [11] uses a hardware accelerator and single instruction multiple data (SIMD) instructions, resulting in increased throughput, reduced area and energy per pixel. Nevertheless, our parallel slice decoder achieves 2.6× higher throughput. Moreover, our designs achieve 3.4× higher throughput and 21× lower energy per pixel than a HEVC decoder [8] on an Intel i7-3720QM processor. Despite lower throughput, our designs achieve up to 8.4× and 8.5× lower energy per pixel than a HEVC decoder [12] on a 6-core Intel i7 E5-1650 processor and a Versatile Video Coding (VVC) decoder [18] on an Intel i9-9980HK processor, respectively.

The proposed DSC decoders support higher resolutions such as 4K and 8K UHD without any design modifications. The size of line buffer, pixel buffer, and rate buffer are related to picture width; therefore, DSC requires larger buffers for higher resolutions. In our designs, all three buffers use 64-KB memory modules, which are sufficient to support 8K. In addition, the parallel slice decoder is fully scalable to achieve higher throughput by increasing the number of parallel modified slice decoders. The throughput and energy efficiency for 4K and 8K are expected to be similar to that for 1080p.

VI. Conclusion

This brief presents two software VESA Display Stream Compression (DSC) decoder designs for many-core processor arrays. The slice decoder design exploits fine-grained task-level parallelism of the DSC decoding algorithm. Performance and area are optimized by minimizing the latency of feedback loops caused by data dependencies. Area is further reduced by fitting more work to each processor in non-critical paths, resulting in smaller number of processors. Furthermore, the scalable parallel slice decoder design leverages slice-level parallelism and results show that by using four modified slice decoders, it achieves 47.9–95.6 fps at 1080p, which is sufficient for most real-time applications. The proposed decoders also support 4K and 8K videos and the parallel slice decoder is fully scalable to achieve higher throughput by processing more slices in parallel.

References

[1] *VESA Display Stream Compression (DSC) Standard v1.2a*, Video Electron. Stand. Assoc. Stand., Jan. 2017. [Online]. Available: http://vesa.org

[2] F. G. Walls and A. S. MacInnis, "VESA display stream compression for television and cinema applications," *IEEE Trans. Emerg. Sel. Topics Circuits Syst.*, vol. 6, no. 4, pp. 460–470, Dec. 2016.

[3] R. S. Allison *et al.*, "Large scale subjective evaluation of display stream compression," in *SID Symp. Dig. Tech. Papers*, vol. 48, 2017, pp. 1101–1104.

[4] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 7, pp. 560–576, Jul. 2003.

[5] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand, "Overview of the high efficiency video coding (HEVC) standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1649–1668, Dec. 2012.

[6] J. Ohm, G. J. Sullivan, H. Schwarz, T. K. Tan, and T. Wiegand, "Comparison of the coding efficiency of video coding standards—Including high efficiency video coding (HEVC)," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1669–1684, Dec. 2012.

[7] M. Horowitz, A. Joch, F. Kossentini, and A. Hallapuro, "H.264/AVC baseline profile decoder complexity analysis," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 7, pp. 704–716, Jul. 2003.

[8] F. Bossen, B. Bross, K. Suhring, and D. Flynn, "HEVC complexity and implementation analysis," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1685–1696, Dec. 2012.

[9] J. Xu, R. Joshi, and R. A. Cohen, "Overview of the emerging HEVC screen content coding extension," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 26, no. 1, pp. 50–62, Jan. 2016.

[10] W.-H. Peng *et al.*, "Overview of screen content video coding: Technologies, standards, and beyond," *IEEE Trans. Emerg. Sel. Topics Circuits Syst.*, vol. 6, no. 4, pp. 393–408, Dec. 2016.

[11] S. Zhu, Z. Yu, S. Cui, Z. Yu, and X. Zeng, "H.264 video parallel decoder on a 24-core processor," in *Proc. IEEE 10th Int. Conf. ASIC*, Shenzhen, China, 2013, pp. 1–4.

[12] W. Hamidouche, M. Raulet, and O. Déforges, "4K real-time and parallel software video decoder for multilayer HEVC extensions," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 26, no. 1, pp. 169–180, Jan. 2016.

[13] S. Wu and B. M. Baas, "A low-cost slice interleaving DSC decoder architecture for real-time 8K video decoding," in *Proc. IEEE 61st Int. Midwest Symp. Circuits Syst. (MWCAS)*, Windsor, ON, Canada, Aug. 2018, pp. 364–367.

[14] S. Wu, S. Gutgutia, M. Alioto, and B. Baas, "Display stream compression encoder architectures for real-time 4K and 8K video encoding," in *Proc. 52nd Asilomar Conf. Signals Syst., Comput. (ACSSC)*, Pacific Grove, CA, USA, Oct. 2018, pp. 251–255.

[15] S. W. Kim, S. Park, J. Jun, and Y. Han, "Design and implementation of display stream compression decoder with line buffer optimization," *IEEE Trans. Consum. Electron.*, vol. 65, no. 3, pp. 322–328, Aug. 2019.

[16] B. Bohnenstiehl *et al.*, "KiloCore: A 32-nm 1000-processor computational array," *IEEE J. Solid-State Circuits*, vol. 52, no. 4, pp. 891–902, Apr. 2017.

[17] S. Wu and B. M. Baas, "Indexed color history many-core engines for display stream compression decoders," in *Proc. 27th IEEE Int. Conf. Electron. Circuits Syst. (ICECS)*, Glasgow, U.K., Nov. 2020, pp. 1–4.

[18] A. Wieckowski *et al.*, "Towards a live software decoder implementation for the upcoming versatile video coding (VVC) codec," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Abu Dhabi, UAE, Oct. 2020, pp. 3124–3128.

[19] W. M. Holt, "Moore's law: A path going forward," in *IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers*, San Francisco, CA, USA, 2016, pp. 8–13.

[20] M. Butler, "'Bulldozer' a new approach to multithreaded compute performance," in *Proc. HotChips Symp. High-Perform. Chips*, Stanford, CA, USA, 2010, pp. 1–17.

[21] B. Bohnenstiehl *et al.*, "KiloCore: A fine-grained 1,000-processor array for task-parallel applications," *IEEE Micro*, vol. 37, no. 2, pp. 63–69, Mar./Apr. 2017.