OPIR Video Preprocessing and Compression for On-Board Aerospace Computing

Eric Shea, Alan George
NSF Center for High-Performance Reconfigurable Computing (CHREC)

ECE Dept., University of Pittsburgh
Room 1238D, Benedum Hall
Pittsburgh, PA 15261
{eric.shea, alan.george}@pitt.edu

Abstract – **Increasing bit-depth of new image sensors presents many challenges on resource-limited, on-board processors in aerospace. This paper provides new results, analysis, and insight with our novel methods for preprocessing with compression of an Overhead Persistent InfraRed (OPIR) image sensor on embedded processors including Xilinx Zynq-7020 and Amlogic S905.**

I. Introduction

The increasing bit-depth on next-generation image sensors presents many unique challenges for video preprocessing and compression with on-board processors in aerospace computing. The naturally limiting resources, such as memory and downlink bandwidth, require more efficient methods to achieve faster execution of preprocessing and higher compression ratios (CRs). However, trying to accomplish these tasks will create a trade-off between video quality and data throughput that the system designer will need to consider based on their requirements and the resources available on their system. The underlying goal is to perform preprocessing, compression, and downlink faster than it would be to downlink the raw data from an Overhead Persistent InfraRed (OPIR) image sensor. Previous research proved that we can do exactly that with one of our novel preprocessing methods, Bit-Stream Splitting (BSS), at bandwidths less than 6 Mbps. The scope of this research was limited though since it did not test our other preprocessing method, SuperFrame. This paper provides an analysis of SuperFrame alone and in combination with BSS on different CPUs to achieve faster downlink of sensor data as opposed to downlinking the raw data for bandwidths typically available for on-board systems. Based on the data and analysis presented, a designer will be able to determine which preprocessing method, architecture, and encoder to use.

As video systems become more mainstream for spacecraft, drones, and other unmanned aerial vehicles, the image sensors are getting more diverse and complex as the demand for better resolution, higher frame rates, and higher bit-depth increases. There are many dependent variables to consider when designing to achieve maximum performance of a real-time video system such as the preprocessing method, architecture, bit-depth of the image sensor, and encoder. For example, our two preprocessing methods effect achievable compression ratios and cause a small delay in the video data. The CPU architecture, memory, and clock frequency effects how quickly preprocessing and compression can be executed. The bit-depth of the image sensor can limit us to a select few encoders since most can only handle standard 8- and 16-bit data types. The encoder effects execution time, compression ratio, ability to be parallelized based on its multithreading capabilities, bit-depth, input pixel format, and whether we can perform lossless and/or lossy compression.

As the bit-depth of image sensors increases, such as the 14-bit OPIR sensor, the image data surpasses most of the standard 8-bit codecs. Only a few codecs can compress anything higher than 8-bit, such as PNG [3], JPEG-LS [4], FFV1 [5], FFVhuff [6], and JPEG2000 [7]. Furthermore, one of these can only do lossless encoding and can only compress a single frame at a time. Since popular codecs such as x264 (an open-source version of h.264), MJPEG, and VP8/9 can only process 8-bit data, we must perform preprocessing on the original video data since it is 14 bits per pixel.

This paper includes background and related work to give insight into our novel methodology and provide results from previous work. It explains our two preprocessing methods and three compression methods that we use to increase the compression ratio and speed. In addition, it includes the experimental setup, which details the two different CPU architectures used and the OPIR sensor. We outline different performance metrics used to analyze the data and finally, the results section provides a comparative analysis between the performance of each preprocessing method, CPU architecture, and image encoder.

II. Background and Related Work

Extensive research has been done to achieve higher CRs with low root-mean-square error (RMSE) for OPIR sensors. RMSE is a metric we use to compare the original video to the decompressed video. Without using preprocessing, we were only able to achieve a CR between 1.0 to 1.9 using JPEG2000, JPEG-LS, FFV1, and FFVhuff. These CRs are not acceptable for real-time systems, nor worth the effort to perform compression. As a result, we developed two novel preprocessing methods, BSS and SuperFrame, to help

978-1-5386-3200-0/17/$31.00 ©2017 IEEE

achieve a more worthwhile CR. We found that, using BSS with x264 encoder, we could obtain a CR of ~25 and an RMSE of ~15[1]. Using SuperFrame alone with an image resolution of 256×256×4200, we could achieve an average lossless CR of ~2.44 using JPEG2000, PNG, and JPEG-LS. PNG had the best lossless CR of 3.57 for SuperFrame. A combination of BSS and SuperFrame was also analyzed where it performed better than SuperFrame alone for CRs higher than 20. Other preprocessing methods such as filtering, factoring, and region-of-interest (ROI) were also studied, but they did not provide as good of performance compared to BSS and SuperFrame, so they were not extensively studied.

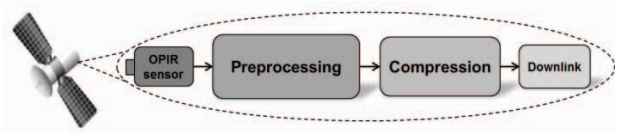


Fig. 1. Block Diagram of our Methodology

Our novel methodology is shown in Fig. 1 to illustrate our intent to do on-board processing of video data from an OPIR sensor. The block diagram consists of preprocessing using our novel methods such as BSS and SuperFrame, lossless or lossy compression, and downlink of the compressed data to a ground station. The decompression performance of the video data was not analyzed for this study since it will be done on a ground-based system.

Once we determined that BSS and SuperFrame were the most effective preprocessing methods, we wanted to see how long it took to execute them on resource-limited platforms studied in [2]. Moreover, three progressively more aggressive compression methods, illustrated in Fig. 2, were developed and used with BSS to see the speedups achieved compared to a serial Baseline Method. Using the Parallelized Method in Fig. 2, we could decrease the *execution time* (time it takes to perform preprocessing and compression) by 2× using two threads compared to the serial Baseline method which used a single thread. The Augmented Parallelized Method further decreased the execution time by offering a 3.4× average speedup over the serial Baseline Method.

We also compared the execution time between 8- and 16-bit encoders such as x264, PNG, JPEG-LS, FFV1, FFVhuff, and FFV1 Version 3. FFVhuff provided the fastest execution time which was due to inefficient compression of OPIR video, while PNG and FFV1 executed the slowest for the 16-bit encoders. We found that using BSS with the 8-bit encoder, x264, had a much slower execution time due to having to account for the extra time needed to carry out BSS. However, it offered the highest CR compared to the other encoders used.

The overall goal was to show that our methods will allow for faster downlink of the sensor data as opposed to downlinking the raw data for bandwidths typically available for on-board systems. We could downlink the BSS plus x264 data faster than downlinking the RAW data for bandwidths lower than 6 Mbps on the P5040, Amlogic S905, and Xilinx Zynq-7020. The RAD5545 was not able to preprocess, compress, and downlink the data faster than downlinking the RAW data for all bandwidths due to the low clock frequency of a radiation-hardened processor.

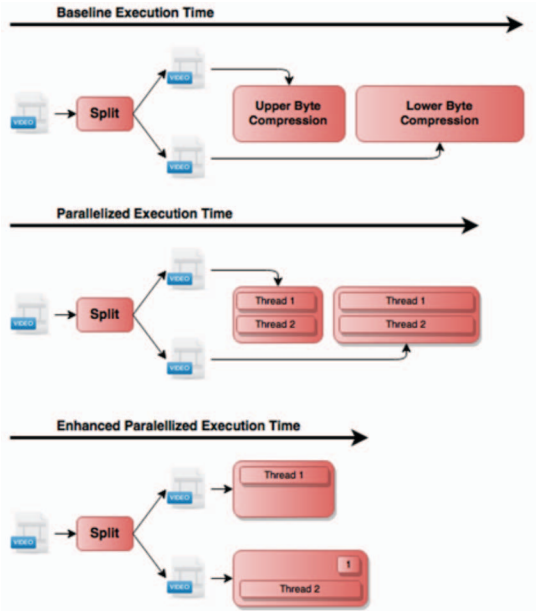


Fig. 2. BSS Compression Methods

III. Experimental Setup

BSS and SuperFrame, and compression, using FFmpeg's open-source compression library[6],were executed using ODROID-C2 and Avnet ZedBoard development platforms because they are part of NASA's High-Performance Spaceflight Computing (HPSC) project and the CHREC Space Processor (CSP)[8], respectively. Their system specifications are shown in Table 1.

Table 1: Targeted System Specifications

Board	Chipset	Architecture	Frequency (GHz)
ODROID-C2	Amlogic S905	4× ARM Cortex- A53	1.500
Avnet Zedboard	Xilinx Zynq-7020 SoC XC7Z020	2× ARM Cortx-A9	0.766

For our analysis, the Air Force Research Laboratory (AFRL) provided us with three video files containing simulated video data using landsat imagery for backgrounds. A single frame from each video file is illustrated in Fig. 3. Each video file has roughly 4200 frames, an image resolution of 256×256 pixels, a bit-depth of 14, and a grayscale pixel format. These specifications were chosen to mimic commercial, scientific-grade cameras currently available for lab use. The 14-bit data is housed in a 16-bit data container within the video file, which allows us to leverage standard 16-bit encoders.

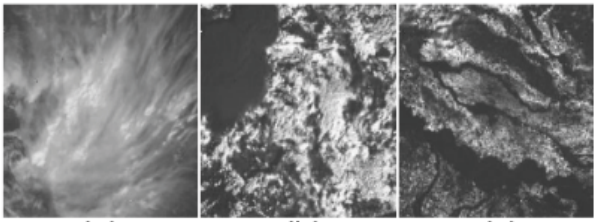


Fig. 3. Representative OPIR sample frames from simulated 14-bit video test set for varying cloud cover situations: (a) Cloud001, (b) Cloud002, and (c) NoCloud001

Our SuperFrame method can leverage various image encoders since it is essentially a very large image. In this study, we analyze libopenJPEG (open-source version of JPEG2000), JPEG-2000, JPEG-LS, and PNG. All four of these encoders are able to support an 8- and 16-bit bit depth. To take advantage of the 8-bit side of the image encoders we performed BSS on the original, high bit-depth video file to create one file containing only the lower bytes of each pixel and the other file containing the higher bytes of each pixel. Then, we executed SuperFrame on both video files. Since we used BSS, it allowed us to perform each of the three Compression Methods shown in Fig. 2.

IV. Performance Evaluation Metrics

The same metrics that were used in [2] are also used in this study. Using SuperFrame alone, we are not able to use the Compression Methods in Fig. 1, but when we perform BSS in combination with SuperFrame, we are able to leverage all three methods. The main way to evaluate the compression speed of an encoder is to measure the amount of time it takes to complete the encode processing. In our experiment, we used the Linux *time()* function to measure the encoder execution time. In addition to the encoder execution time, we also measured the execution time for any preprocessing on the input data before beginning the encoding, which includes the execution time for BSS.

Total run-time (Equation 1) includes the execution time for preprocessing and compression and the transfer time for downlink based upon the available bandwidth. We do not consider decoding time in this metric because a typical application would operate the decoding process on high-performance, ground-based servers.

$$\textit{Total Run-Time} = \textit{Execution Time} + \textit{Transfer Time} \qquad (1)$$

V. SuperFrame Results

A. 16-Bit Lossless Image Encoders

In previous research, we performed our SuperFrame method on a desktop computer with high memory and computational power with respect to on-board processing systems. We could use a significantly larger SuperFrame size of 256×256×4200 (H × W × Frames), however, this size is not achievable on the Amlogic S905 and Xilinx Zynq-7020 due to the limited memory. We found that the maximum SuperFrame size for each platform was 256×256×350 for lossless compression. At this size, a lossless CR of ~3.5 was achieved for all encoders. This maximum SuperFrame size achieved a 2.3× greater CR compared to doing standard frame-by-frame compression, which is shown in Fig. 4.

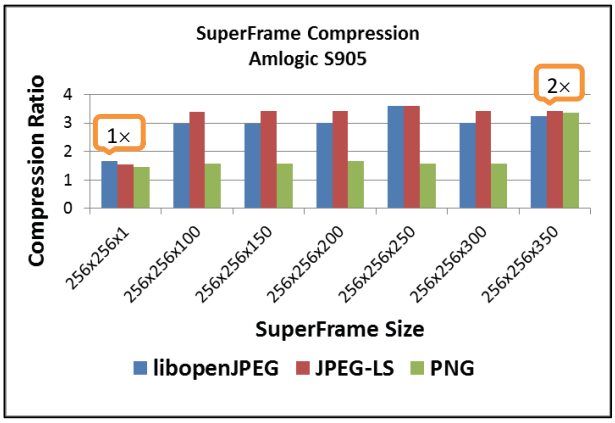


Fig. 4. Lossless Compression Ratios for Varying SuperFrame Sizes

As mentioned above, the execution time is the time it takes for preprocessing and compression. Fig. 5 shows the execution times for libopenJPEG, JPEG-LS, and PNG on the Amlogic S905. JPEG-LS and libopenJPEG offered the fastest execution with a 20.9× and 2.6× speedup over PNG, respectively. At higher SuperFrame sizes, the execution time is much slower due to the increased size needed for the memory buffer, resulting in reduced efficiency between the shared memory of the system and application. This is important to consider as we try to minimize the execution time as much as possible to achieve a faster downlink of video data. The execution times of libopenJPEG and JPEG-LS were, on average, 1.7× and 1.8× faster using SuperFrame, respectively. However, for PNG, the execution time was 2.1× slower using SuperFrame.

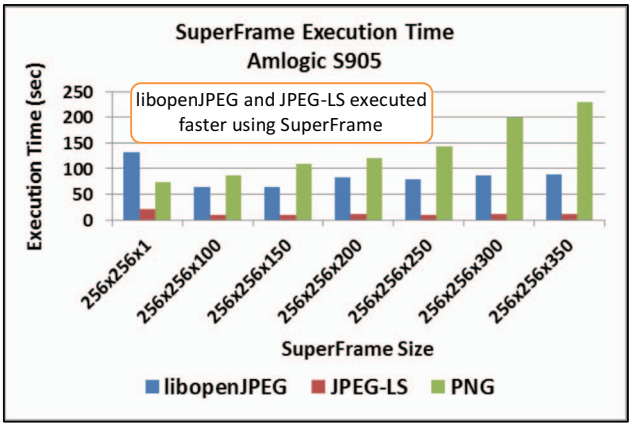


Fig. 5. Execution Time for Lossless 16-Bit Encoders libopenJPEG, JPEG-LS, and PNG on Amlogic S905

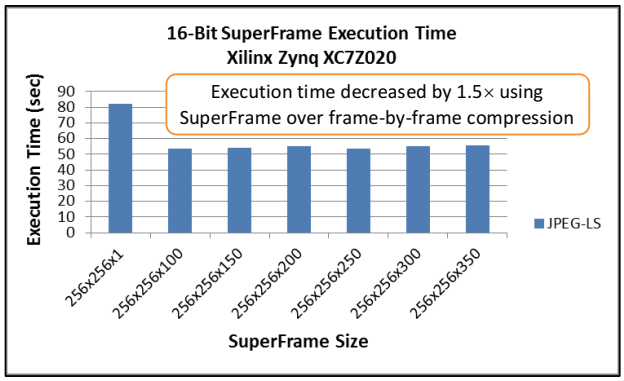


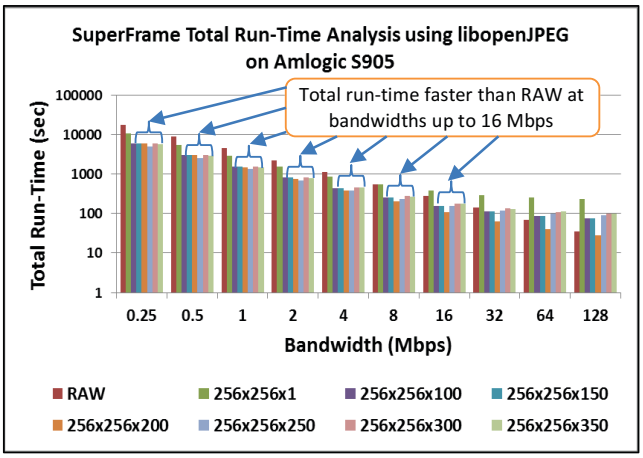
Fig. 6. Execution Time for Lossless 16-Bit JPEG-LS on Xilinx Zynq XC7Z020

The execution time for JPEG-LS on the Xilinx Zynq XC7Z020 chipset is shown in Fig. 6. Similar to the execution time of JPEG-LS on Amlogic S905, the execution time decreased by an average of 1.5× for increasing SuperFrame sizes. The Amlogic S905 offered a 3.9× and 4.5× speedup over the Xilinx Zynq XC7Z020 for frame-by-frame and SuperFrame compression, respectively. This was due to the higher clock frequency of the Amlogic S905.

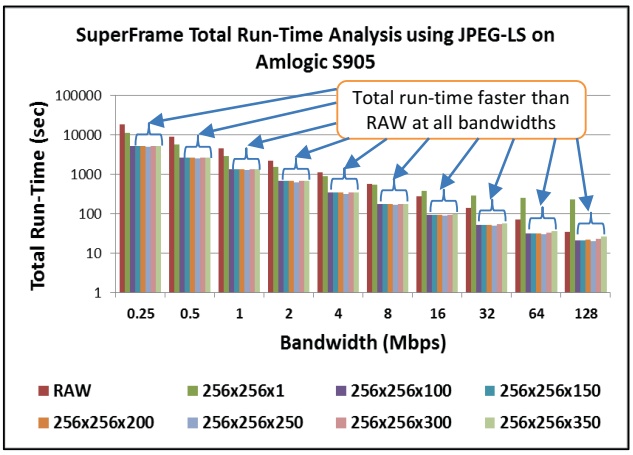
FFmpeg does not include libopenJPEG and PNG by default so their external libraries had to be cross-compiled for the Xilinx Zynq chipset. We were unsuccessful in doing this so only the execution time for JPEG-LS was analyzed.

B. Total Run-Time Analysis for 16-Bit Lossless Encoders

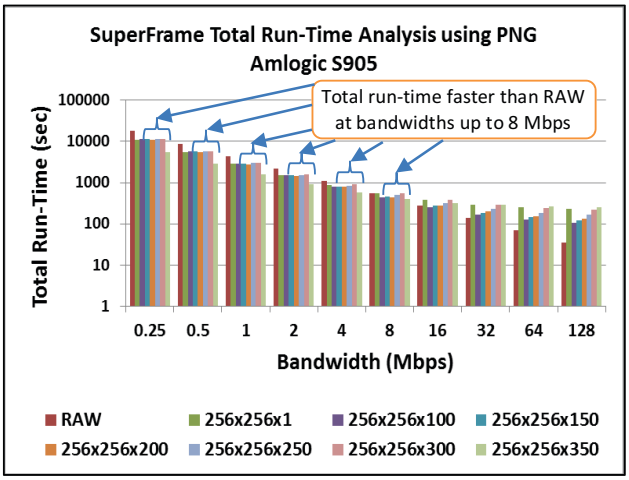
Similar to our previous research in [2], we performed a total run-time analysis, shown in Figs. 7 and 8, to show that we can perform preprocessing, compression, and downlink of the video data faster than downlinking the raw data to achieve real-time video. We found that we could achieve this at bandwidths less than 16Mbps for libopenJPEG, all bandwidths for JPEG-LS, and bandwidths less than 8 Mbps for PNG on the Amlogic S905. On the Xilinx Zynq XC7Z020 in Fig. 8, we found that we could achieve faster downlink for bandwidths less than 32 Mbps. However, bandwidths above 16Mbps are uncommon for on-board processing systems in aerospace. A SuperFrame size of 256×256×350 and 256×256×200 provided the fastest total run-time offering a 3.2× speedup over RAW. The standard frame-by-frame compression provided the slowest total run-time, but still provided a 1.7× speedup over RAW. Overall, JPEG-LS had the fastest total run-time compared to PNG and libopenJPEG for all SuperFrame sizes since it provided the best compression ratio as shown in Fig. 4.



(a)



(b)



(c)

Fig. 7. Total Run-Time Analysis for Varying SuperFrame Sizes using (a) libopenJPEG, (b) JPEG-LS, and (c) PNG on Amlogic S905

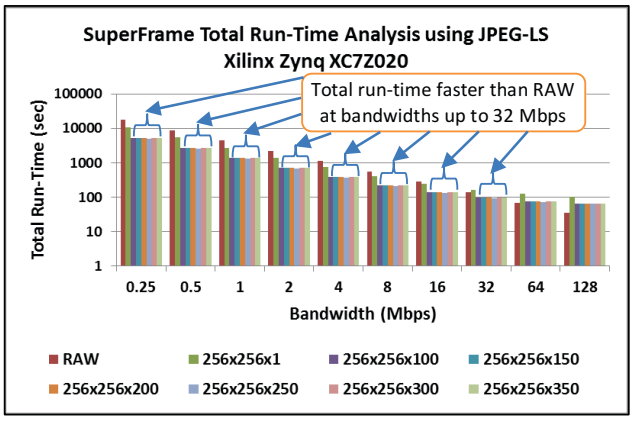


Fig. 8. Total Run-Time Analysis for Varying SuperFrame Sizes using JPEG-LS on Xilinx Zynq XC7Z020

C. 16-bit Lossy Image Encoders

The performance in terms of compression ratio and execution time of 16-bit lossy image encoders were analyzed. JPEG-LS and PNG are lossless only, so libopenJPEG was solely evaluated. The lossy compression ratios are shown in Fig. 9 and the execution times are shown in Fig. 10 using libopenJPEG for SuperFrame sizes of 256×256×100 and 256×256×200. As the compression level increases, the compression ratio exponentially increases. A compression level of four provides a good tradeoff between compression ratio and RMSE.

Due to the addition of the quantization step needed to perform lossy compression, the execution time increases by 1.4× and 1.3× for a SuperFrame size of 256×256×100 and 256×256×200, respectively, as shown in Fig. 10. However, due to the slight increase in execution time of 25 seconds, we would still be able to preprocess, compress, and downlink the video data faster than RAW for bandwidths less than 16 Mbps. Based on these results, a system designer will need to balance between lossless and lossy compression ratios, video quality, and execution time.

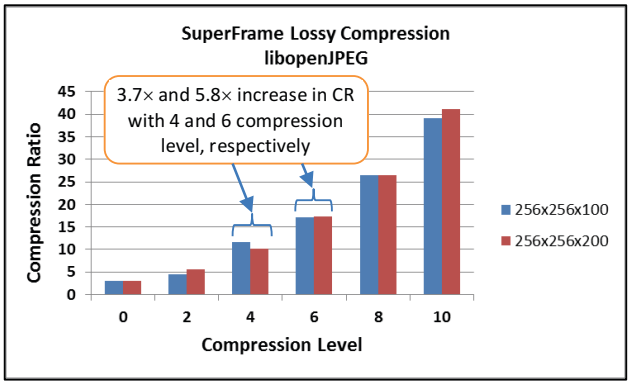


Fig. 9. Lossy Compression Ratios for Varying SuperFrame sizes using libopenJPEG

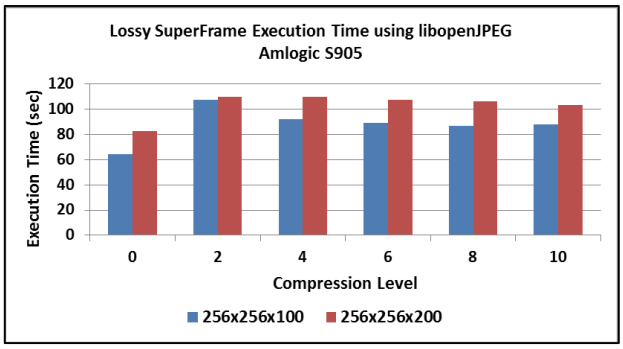


Fig. 10. Execution Time using Lossy 16-bit libopenJPEG

VI. BSS WITH SUPERFRAME RESULTS

Next, we performed BSS in combination with SuperFrame to leverage the 8-bit side of JPEG2000, JPEG-LS, and PNG. We achieved a 1.2× CR for JPEG2000, while JPEG-LS had roughly the same CR, and PNG decreased by 1.6× compared to the CRs obtained from using SuperFrame alone with a 16-bit encoder. These results are shown in Fig. 11.

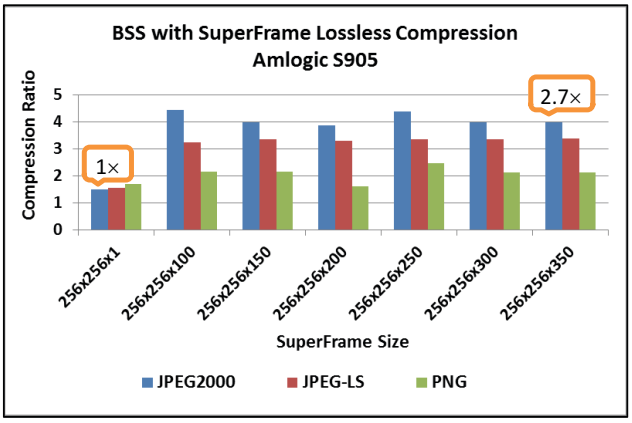


Fig. 11. Lossless Compression Ratios using BSS with SuperFrame on Amlogic S905

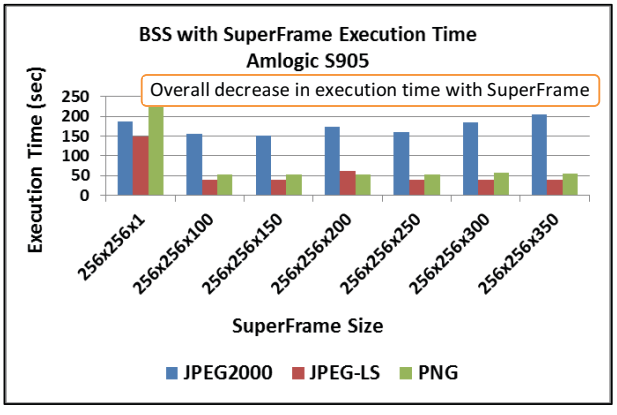


Fig. 12. BSS with SuperFrame Lossless Execution Time using Amlogic S905

The execution times for varying SuperFrame sizes using BSS in combination with SuperFrame on the Amlogic S905 are shown in Fig. 12. Overall, the execution time decreases with a higher SuperFrame size by an average of 1.1×, 3.4×, and 4.3× for JPEG2000, JPEG-LS, and PNG, respectively. Compared to their 16-bit counterpart, JPEG2000, JPEG-LS, and PNG executed 1.4×, 7.1×, and 3.1× slower for frame-by-frame compression, respectively, due to the additional time needed to execute BSS. Leveraging 8-bit SuperFrame for compression JPEG2000 and JPEG-LS executed 2.2× and 3.6× slower, respectively, while PNG executed 2.8× faster.

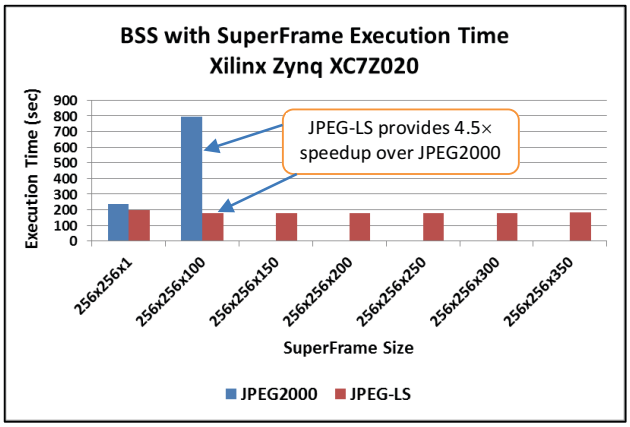
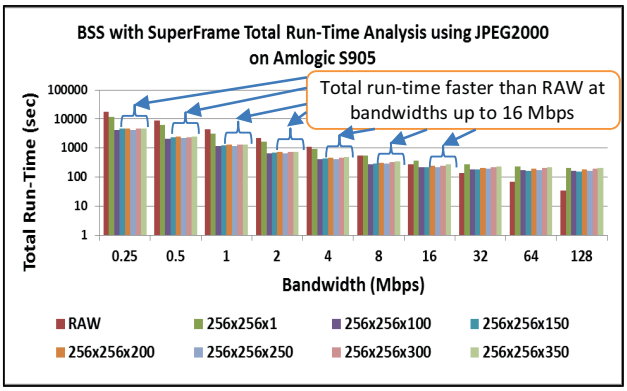


Fig. 13. BSS with SuperFrame Execution Time using Xilinx Zynq XC7Z020

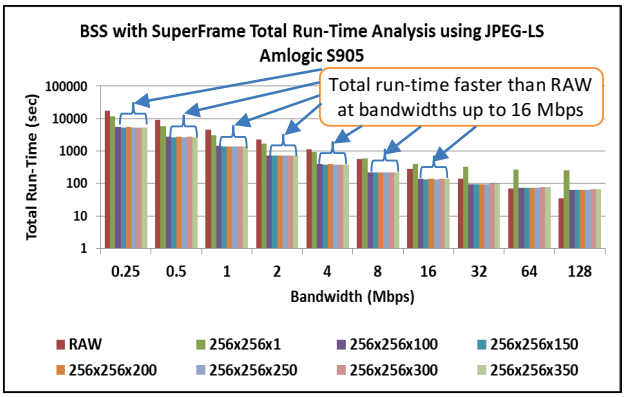
The execution times for varying SuperFrame sizes using BSS in combination with SuperFrame on the Xilinx Zynq XC7Z020 are shown in Fig. 13. The execution time for JPEG2000 could only be found for a SuperFrame size up to 256×256×100 due to the memory allocation restrictions. At a SuperFrame size of 256×256×100, JPEG-LS had a 4.5× speedup over JPEG2000. If a system designer were to choose JPEG2000 as the encoder and a SuperFrame size of 256×256×100, they would get a good CR as shown in Fig. 11, but would have to consider the increased latency of it. JPEG2000 and JPEG-LS executed more efficiently on the Amlogic S905 by offering a 5.1× and a 1.9× speedup over the Xilinx Zynq XC7Z020.

A. Total Run-Time Analysis for 8-bit Lossless Encoders

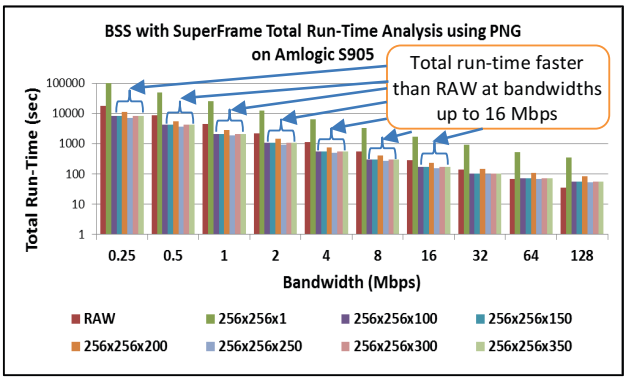
A total run-time analysis was also performed for 8-bit lossless encoders on the Amlogic S905 as shown in Fig. 14. We found that we could achieve preprocessing, compression, and downlink faster than RAW at bandwidths less than 16Mbps for JPEG2000, bandwidths less than 32 Mbps for JPEG-LS, and bandwidths less than 16 Mbps for PNG on the Amlogic S905. Due to the increased efficiency of PNG's 8-bit side, we could perform our methodology up to 16 Mbps compared to just 8 Mbps of its 16-bit side.



(a)



(b)



(c)

Fig. 14. BSS with SuperFrame Total Run-Time Analysis for Varying SuperFrame Sizes using (a) JPEG2000, (b) JPEG-LS, and (c) PNG on Amlogic S905

VII. Conclusions

The analysis presented in this paper provides beneficial insight for using CPUs to maximize performance for video preprocessing and compression. The experiment provided the speedups achieved for executing our novel preprocessing methods, compression, and downlink across all platforms which allows a designer to successfully choose which preprocessing method, architecture, and encoder to use for an OPIR image sensor based on their system specifications. We found that due to the resource

limitations of the platforms used, we were not able to use a maximum SuperFrame size of 256×256×4200 used in previous research. The maximum SuperFrame size achieved for lossless compression was 256×256×350 on the ODROID-C2 and Avnet ZedBoard development platforms, but it still showed worthy compression ratios for libopenJPEG, JPEG-LS, and PNG compared to doing standard frame-by-frame compression. It was found that performing lossy compression resulted in an even smaller SuperFrame size of 256×256×250 due to the extra memory needed to perform the quantization step. The execution time of each image encoder was analyzed and it was found that JPEG-LS offered the fastest 16-bit execution time compared to libopenJPEG, JPEG2000, and PNG, while PNG offered the fastest 8-bit execution time. We were able to show that our preprocessing and compression methods allowed for faster downlink of OPIR data as opposed to downlinking the raw data for bandwidths typically available for on-board systems.

Future work will be to make the system a more heterogeneous architecture. This will include integrating an FPGA with a hard-core CPU, and a GPU into the system to perform video preprocessing and compression; the FPGA will perform the preprocessing through an optimized pipeline, the CPU will package the data into a raw video file which will then be sent to the GPU to perform the compression to take advantage of the GPU's data-parallel architecture.

ACKNOWLEDGEMENTS

This work was supported by the CHREC Center members and by the I/UCRC Program of the National Science Foundation under Grant No. IIP-1161022.

REFERENCES

[1] A. Ho, A. George, A. Gordon-Ross, "Improving Compression Ratios for High Bit-Depth Grayscale Video Formats," Proc. of IEEE Aerospace Conference, Big Sky, MT, Mar. 5-12, 2016.

[2] A. Ho, E. Shea, A. George, A. Gordon-Ross, "Comparative Analysis of Parallel OPIR Compression on Space Processors," Proc. of IEEE Aerospace Conference, Big Sky, MT, Mar. 4-11, 2017.

[3] W3C, *Portable Network Graphics (PNG) Specification (Second Edition)*, November 2003

[4] Information Technology-Lossless and near-lossless compression of continuous-tone images-Baseline. International Telecommunication Union (ITU-T Recommendation T.87). ISO/IEC 14495-1, 1998.

[5] FFV1 Video Codec Specification [Online]. Available: http://ffmpeg.org/~michael/ffv1.html

[6] FFMPEG [Online]. Available: http://www.ffmpeg.org

[7] ISO/IEC 15 441-1: Information Technology-JPEG 2000 Image Coding System-Part 1: Core Coding System, 2000.

[8] C. Wilson, J. Urriste, P. Gauvin, J. Stewart, A. George, H. Lam, T. Flatley, G. Crum, M. Wirthlin, "CHREC Space Processor (CSP): A Broad Vision for Hybrid Space Computing," Proc. of 3rd International Workshop on LunarCubes, Palo Alto, CA, Nov. 13-15, 2013.

BIOGRAPHY

Eric Shea *received his B.S degree in EE from the University of Florida. He is an M.S. student and a research assistant in the on-board processing group of the NSF CHREC Center at the University of Pittsburgh.*

Alan George *is Professor of ECE at the University of Pittsburgh, where he serves as serve as Ruth and Howard Mickle Endowed Chair and the Department Chair of Electrical and Computer Engineering in the Swanson School of Engineering at Pitt. He also serves as the Director of the NSF Center for High-performance Reconfigurable Computing (CHREC). He received the B.S. degree in CS and M.S. in ECE from the University of Central Florida, and the Ph.D. in CS from the Florida State University. Dr. George's research interests focus upon high-performance architectures, networks, systems, services, and apps for reconfigurable, parallel, distributed, and fault-tolerant computing. He is a Fellow of the IEEE.*