Comparative Analysis of Parallel OPIR Compression on Space Processors

An Ho[1], Eric Shea[2], Alan George[1,2], Ann Gordon-Ross[1]
NSF Center for High-Performance Reconfigurable Computing (CHREC)

[1]ECE Dept., University of Florida
Room 327, Larsen Hall
Gainesville, FL 32611
{an,ann}@chrec.org

[2]ECE Dept., University of Pittsburgh
Room 1238D, Benedum Hall
Pittsburgh, PA 15261
{george,eshea}@chrec.org

***Abstract* – Requirements for higher video quality in space applications continuously calls for increased resolution in imaging sensors, higher bit-depth codecs, more creative solutions for preprocessing and compression techniques, and faster, yet resilient, space-grade platforms. Understanding how these variables interact and affect each other on different platforms is crucial in system development when trying to meet requirements and constraints, such as compression speed, compression ratio (CR), image quality, bandwidth, etc. To analyze this interaction, we present a comparative analysis between compression speed and compression ratio using serial and parallel compression codes on different platforms and architectures, focusing upon video data from overhead-persistent infrared (OPIR) sensors on spacecraft. Previous research allowed us to compare CR and image quality with new preprocessing techniques, but it did not evaluate and address the challenges of compression speed on space-grade processors. Performance is critical, since of course the preprocessing and compression codes plus downlink of compressed data must require less total time than downlink of the raw data, in order for compression to be fully effective.**

TABLE OF CONTENTS

1. INTRODUCTION

Increasing demand for higher video quality coupled with limited communication bandwidth, and the need to recreate lossless or nearly lossless images after compression and transmission to ground stations, are pushing existing spacecraft to their limits. Given oftentimes non-standard video formats from specialized sensors, developing entirely new codecs and system designs can be impracticable with limited design time and budget. Processing architecture, codec, type of compression (lossy, lossless), video preprocessing, and compression techniques must all be considered when choosing the most appropriate combination of these variables to meet mission requirements and constraints, such as compression speed, compression ratio (CR, which compares size of compressed data to size of uncompressed data), video and image quality, required bandwidth, etc. Understanding the correlation between these variables (i.e., how they affect each other) is crucial for efficient system development and closer adherence to application and mission needs.

As the resolution (bit-width) of image sensors increases, such as 14-bit, overhead-persistent infrared (OPIR) sensors, 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 [2], JPEG-LS [3], FFV1 [4], FFVhuff [5], and JPEG2000 [6], and some 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, can only process 8-bit data. In prior work [1], we developed and evaluated a preprocessing technique called bit-stream splitting (BSS), which splits the original data file into two 8-bit files, one containing the uppermost bytes and one containing the lowermost bytes, so as to use 8-bit encoding.

Unfortunately, lossless compression results in low CR, and some applications cannot tolerate the data loss incurred due to lossy compression to get higher CR. Thus, to increase CR, we explored other new preprocessing techniques, such as filtering, region of interest (ROI), factoring, and SuperFrame to increase the CR of 14-bit grayscale OPIR videos. Filtering reduces the noise by using Gaussian, Median, Average, and Wiener noise filters. The ROI technique only performs lossless compression on the most important region of the video frame, and performs lossy compression on the remaining frame data. Factoring removes the bits from each pixel value that do not have a large effect on image quality. SuperFrame concatenates many video frames into one large frame, where each row in the SuperFrame represents one frame of the video file. Then, once the preprocessing technique is performed, the new video file can then be compressed and evaluated.

We evaluated these video preprocessing techniques, alone and in combination, to evaluate processing's effects on CR and the resulting video quality, which was measured using

978-1-5090-1613-6/17/$31.00 ©2017 IEEE

the root-mean-square error (RMSE), which is a common metric that measures the difference between the original pixel and the transformed pixel (after encoding and decoding) for each pixel in the video file. Higher RMSE denotes poorer video quality.

Our prior work revealed several insights into the achievable CR with respect to video quality for each encoder and preprocessing technique, however we did not evaluate and address the challenges of compression speed on space-grade processors [1]. These challenges are caused by the computational complexity of the problem as well as platform memory and bandwidth constraints due to the very high frame rate that is being tested, which is a critical factor for designers if the bandwidth and/or compression time are unachievable. To operate with lower bandwidth, designers can compress the video as much as possible while still retaining the desired data integrity. Alternatively, if the platform has sufficient bandwidth and the designer requires (near) real-time video playback, they can use the encoder and preprocessing technique with faster compression time as compared to the time to transmit the raw video data. Our analysis presented in this paper provides insights on compression speed verses data transmission time by exploring fast compression methods.

To increase the compression speed, we developed and tested three progressively more aggressive (with respect to harnessing parallelization) compression methods, a baseline method, a parallelized method, and an enhanced parallelized method, on each platform in Table 1. Since BSS produces higher CRs, all methods use BSS for preprocessing, and we varied the compression method and platform tested. Results for lossy compression demonstrate that, for all compression methods, compression speed decreases as the constant rate factor (CRF) increases. CRF is defined as the quality setting for the x264 encoder, which can be set between 0 and 51 where 0 denotes the best quality and 51 denotes the worst quality. Lossless compression speed increases with each method for decreasing image qualities. The enhanced parallelized method achieved the fastest compression time with a 1.15× speedup over the parallelized method and a 3.56× speedup over the baseline method. This case was not the fastest compression speed, but offered the highest speedup between methods and was achieved on the ARM Cortex-A9 cores in a Xilinx Zynq-7020 featured in our CHREC Space Processor (CSP). Our experiments also revealed that processing resources in platforms tested were insufficient for real-time compression, with the four-core FreeScale P5040 processor (same cores featured in the BAE RAD5545™) achieving 15 frames per second (FPS) and the ARM cores of the Zynq achieving 11 FPS. This outcome suggests that existing CPUs in space computers are highly limited in the frame rates that they can compress and thus alternative architectures and accelerators may be required.

Table 1: Targeted System Specifications

Board	Chipset	Architecture	Frequency (GHz)
NXP P5040DS-PB Eval. Board	Freescale P5040	4 x e5500	2.200
*Extrapolated RAD5545	BAE Systems RAD5545	4 x RAD5500	0.466
ODROID-C2	Amlogic S905	4 x ARM Cortex A53	1.500
Avnet Zedboard	Xilinx Zynq-7000 SoC XC7Z020	2 x ARM Cortex A9	0.766

2. Background And Related Work

Our prior work showed that preprocessing techniques are required to achieve a lossless CR higher than 1.85. This finding does not benefit a designer who is trying to meet a low bandwidth constraint or, as in our case, a CR of at least 10 and a RMSE less than 15 (we selected these requirements to represent what a typical designer might desire to achieve). We studied the preprocessing techniques both independently and together and, given our requirements, the largest CR being achieved by using a combination of factoring, SuperFrame, and BSS [1].

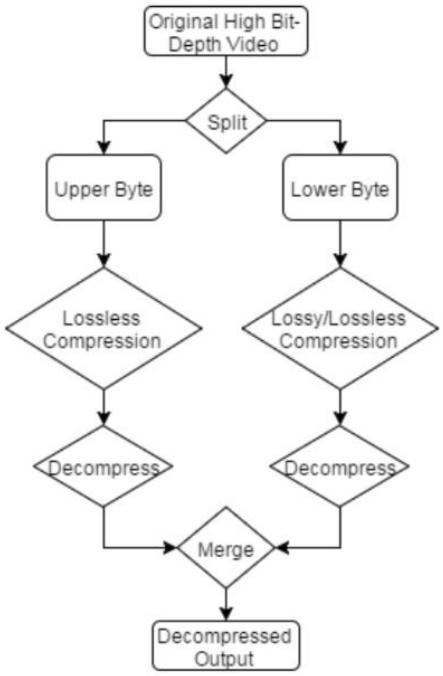


Figure 1: Bit-Stream Splitting (BSS)

Bit-Stream Splitting (BSS)

Since OPIR data is 14-bit, this precludes using 8-bit encoders, however, BSS transforms the data such that the video can be used by any 8-bit encoder. Figure 1 illustrates the BSS process. The original high bit-depth video is split into two partitions, one file containing the pixels' upper bytes and the other containing the pixels' lower bytes. Since more video information is stored in the upper bits, lossless compression is performed on the upper-byte file, and the lower-byte file can either use lossy or lossless compression

depending on the required data integrity. It was found that doing lossy compression on the upper byte resulted in a much higher RMSE. This method is similar to the quantization process in JPEG encoding, wherein low-order bits are considered as 0. Each file is compressed separately and, during decompression, the files are merged back together to recreate the original, high bit-depth video. CR is evaluated by comparing the size of the original video file to the size of both compressed files combined, and RMSE is evaluated by comparing the final decompressed video file to the original video file.

BSS showed the greatest performance benefits as compared to the other preprocessing techniques. Results for different videos with varying cloud cover, ranging from very cloudy to no clouds (Figure 2), achieved CRs of 3.40, 2.45, and 2.60, respectively. Compared to a baseline of lossless compression, BSS increased the CR by 1.74×.

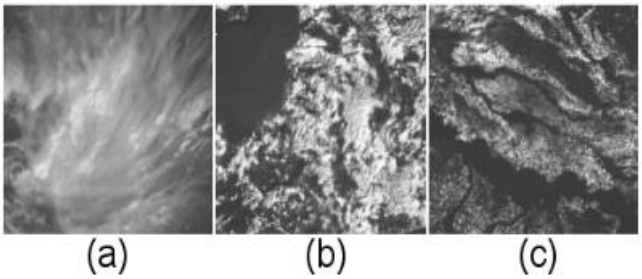


Figure 2: Representative OPIR sample frames from simulated 14-bit video test set for varying cloud cover situations: (a) Cloud001, (b) Cloud002, and (c) NoCloud001

3. EXPERIMENTAL SETUP

Since the analysis in this paper builds on our previous research, we use the same sample data shown in Figure 2 to evaluate our different compression methods. These videos are raw videos from a 14-bit grayscale OPIR sensor. Each file contains ~4000 frames. In order to use these videos with BSS, we increased the bit-width to 16 bits by concatenating two extra zero bits at the most-significant bit positions. OPIR's extremely high frame rate requirement of 300 FPS, as compared to the more common frame rate of 60 or 120 FPS, places extreme pressure on the limited memory and bandwidth available on space-grade platforms.

As cited previously, Table 1 shows the variations of space-grade platforms that we evaluate in this paper. The CHREC Space Processor (CSP) [7] is a platform with promising advancements in space processing capabilities using a commercial processor, the Xilinx Zynq, in a radiation-tolerant system. The BAE Systems RAD5545 is a new radiation-hardened CPU intended for space applications, and it is a target of our work for which we extrapolated by conducting experiments on its commercial counterpart, the FreeScale P5040, and compensating for clock frequency. A target of the future employed here is the multicore CPU being funded for development by the AFRL/NASA High-Performance Spaceflight Computing (HPSC) project. This new radiation-hardened device will feature ARM Cortex-A53 processor cores in the form of interconnected chiplets. CSP is an interesting platform for study since it is a unique combination of commercial and radiation-hardened components plus fault-tolerant computing to garner the high performance and energy-efficiency from commercial with the high reliability of hardened. The RAD5545 is important as a new technology in hardened multicore CPUs. And, the HPSC processor of the future is an interesting target to study since it represents the next-generation space CPU featuring the 64-bit performance of the ARM Cortex-A53.

To gather results, we used the NXP P5040DS-PB Evaluation Board, which uses the P5040 chipset, which is the commercial equivalent of BAE RAD5545. Since we did not have access to the RAD5545, we extrapolated data from P5040 results to approximate achievable performance on the actual RAD5545. For this extrapolation, we used device metrics from [8] to estimate the performance loss going from the commercial equivalent to a radiation-hardened component. Our results revealed that the RAD5545 achieves on average 21.8% of the P5040 performance.

We also used the ODROID-C2 board, which is equipped with the Amlogic S905 device, to evaluate performance of A53 cores, and the Avnet Zedboard to evaluate performance of A9 cores of the CSP, since the Zedboard and CSP use the same processor (Zynq XC7Z020).

We used the open source FFMPEG [5] toolset as the video encoder running on Linux. Encoders tested include PNG, FFV1, FFV1 version 3, FFVhuff, JPEG-LS, and x264. FFV1 version 3 is an experimental implementation of multithreading for FFV1.

4. PERFORMANCE EVALUATION METRICS

To fairly evaluate all aspects of experiments, we consider the execution time for the preprocessing and compression, and the transfer time to downlink the compressed data. This combo enables designers to evaluate both performance of the platform as well as determine the necessary CR required to meet available bandwidth. As CR increases, the compressed file size decreases, necessitating less transmitting bandwidth, which is especially important for applications with lower network bandwidth, such as space applications.

Execution Time

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.

Figure 3 shows our three progressively more aggressive (with respect to harnessing parallelization) compression methods. For all methods, BSS first splits the input data file into the upper- and lower-byte files. The baseline method serially compresses each split file using a single thread. The parallelized method uses the encoder's built-in multithreading capabilities to operate on both split files in parallel. The enhanced parallelized method executes multiple encoder threads in parallel by splitting the platform's resources evenly, and dynamically re-allocates threads to cores after each thread completes execution (not all threads require the same execution time).

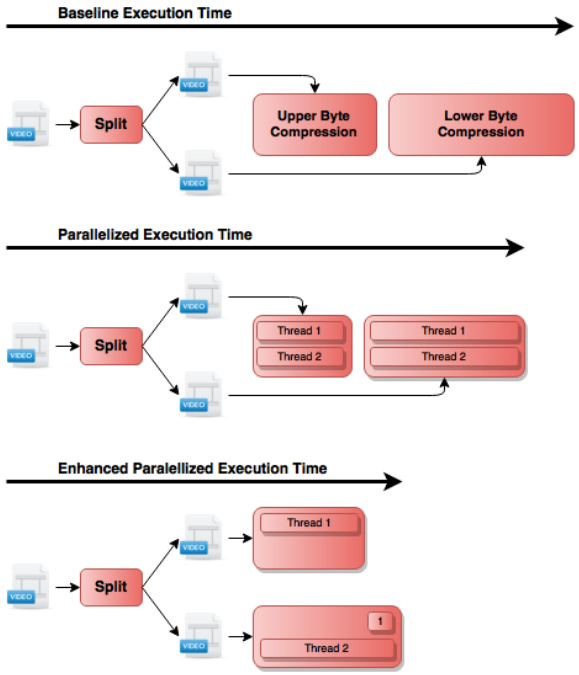


Figure 3: BSS Compression Methods

Total Run Time

Total run time (Equation 1) includes both 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.

$$Total\ Run\ Time = Execution\ Time + Transfer\ Time \quad (1)$$

5. Experimental Results

We gathered results following the procedures described in Section 3 for all three test videos (Figure 2). The results are aggregated and averaged over five experimental runs for each test video.

16-bit Encoding

Figure 4 shows the results from running 16-bit-capable video encoders that support the 14-bit OPIR video format. This experiment enables analysis of the execution time, based on the number of cores used, to see how effectively the target systems can perform for different encoders.

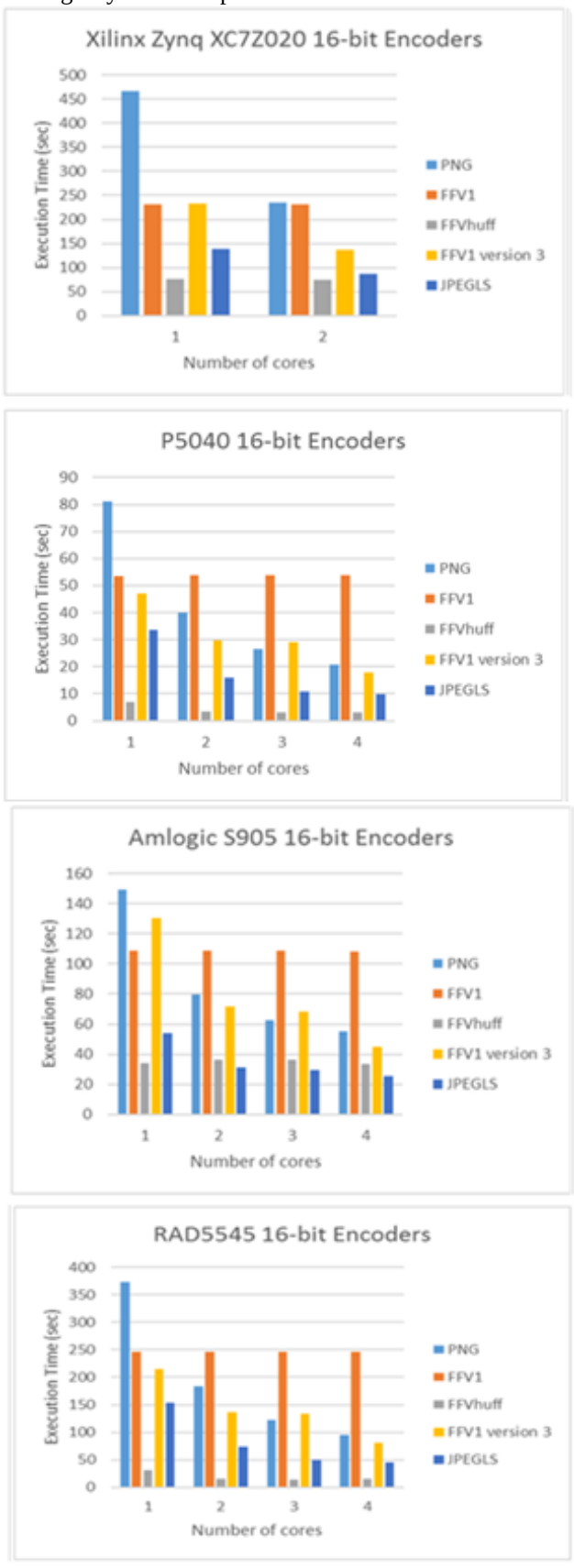


Figure 4: 16-bit encoders performance on (a) Zynq XC7Z020, (b) P5040, (c) S905, and (d) RAD5545

These results show that FFVhuff performed extremely well on all systems. On the Xilinx Zynq, FFVhuff is 1.82 times faster than JPEGLS (the next best performer) using a single thread, and 1.18 times faster than using two threads. Similarly, the P5040 and the extrapolated results for the RAD5545 show that FFVhuff is clearly the fastest encoder on those platforms. On the RAD5545, FFVhuff is projected to be 4.76 times faster than JPEG-LS using a single thread and 3.13 times faster using four threads.

However, FFVhuff does not show promising multithreaded performance, and performed similarly regardless of the number of threads used on the Zynq and S905. However, multithreading did show improvements when moving from using a single thread to using two threads on the P5040 and RAD5545, but additional threads did not improve the compression speed.

Alternatively, FFVhuff was not the best overall performer on the S905. Using a single thread, FFVhuff is faster than JPEG-LS on the S905 but, with multithreading, JPEG-LS achieved the best overall performance on the S905.

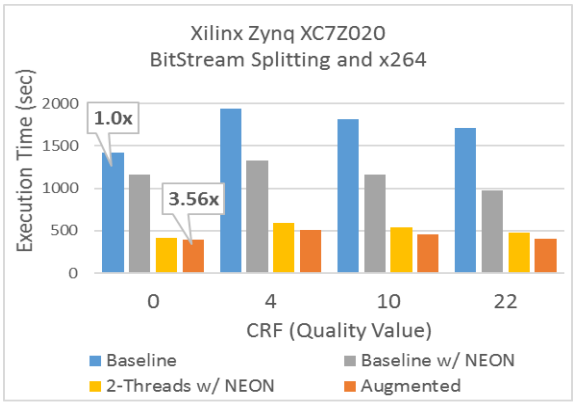
Overall, the results show that the quad-core P5040 is the fastest platform for executing 16-bit encoders, the S905 with quad-core A53 also performs well, and FFVhuff is the fastest encoder out of those tested on a single core.

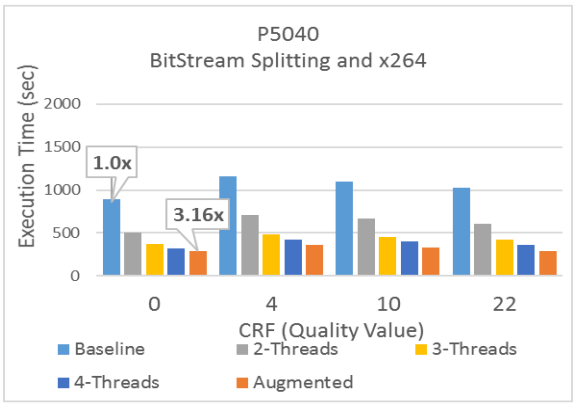
Bit-Stream Splitting (BSS) and x264

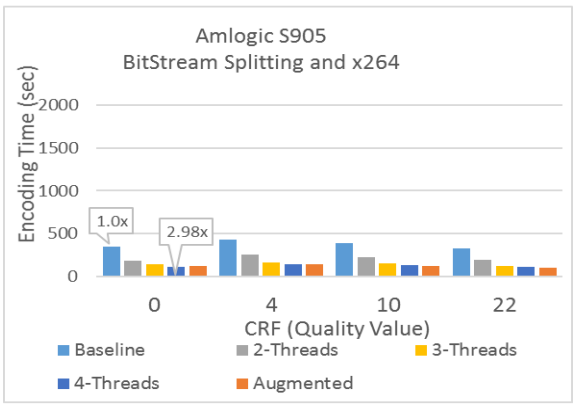
Figure 5 shows execution times for various compression methods described in Section IV on the target platforms. The x-axis indicates the CRF value, which controls the video quality for the x264 encoder. A value of 0 means lossless, and is comparable to using a native 16-bit encoder in terms of video quality. The execution time on the y-axis includes both preprocessing time and compression time.

x264 supports multithreading using pthreads, and this library imparts good execution times that are evident in these results. Execution time decreased by nearly 2× in all systems when using two threads as compared to using a single thread. x264 also performed extremely well on the S905, which is a clear contrast to performance with the 16-bit encoders. The quad-core Amlogic S905 is significantly faster than the other three systems for these cases, whereas the P5040 was the best performer when using 16-bit encoders. We attribute this difference to the x264 being more highly optimized for the ARM Cortex-A53 architecture. The ARM Cortex-A53 has a NEON accelerator unit, which is used for specialized parallel operations. It does speed up the compression process by a significant amount (as shown in Figure 5a for the Xilinx Zynq). The P5040 and the RAD5545 do not have a hardware accelerator unit, and thus they are not as well optimized for x264.

The enhanced parallelized method improved the execution time further by reducing the extra overhead from the starting and stopping of the encoding processing. This







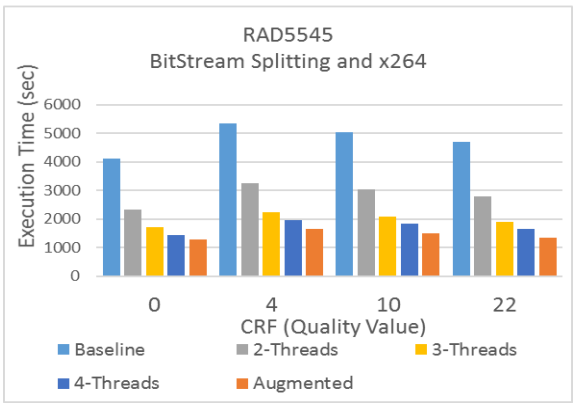


Figure 5: BSS and x264 performance on (a) Zynq XC7Z020, (b) P5040, (c) S905, and (d) RAD5545

method provided an extra 6% to 7% reduction in execution time as compared to the parallelized method.

When comparing to execution times with 16-bit encoders, BSS plus x264 is much slower. On the Zynq, BSS plus x264 is 5.54× slower than FFVhuff, 97.33× slower on P5040 and RAD5545, and 3.18× slower on the S905.

From the results in Figures 4 and 5, we observe that BSS increases the compression time as compared to using 16-bit encoders, but we reiterate that an encoder cannot be evaluated solely based on compression speed, and CR must also be considered.

Total Run Time

Figure 6 shows total run time as a function of bandwidth (in megabits per second, Mbps) and the fastest execution time for each of the encoders and systems. As defined by Equation 1, the portion of total run time for data transmission is calculated by dividing the compressed file size (upper- and lower-byte files summed) by the bandwidth.

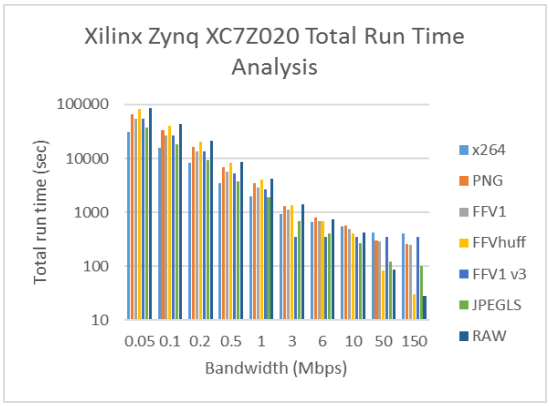
Analysis of the data shows that using BSS plus x264 results in the fastest total run time with a lower bandwidth for the Xilinx Zynq, P5040, and S905. This outcome is not the case for higher bandwidths, achieving 1 Mbps, with JPEGLS being the fastest encoder. This behavior is due to the fact that BSS plus x264 provides the highest CR out of all of the encoding options, which increases the lossless CR by 1.74× as compared to the next best encoding option. Additionally, for certain bandwidths, 1 Mbps in this case, this higher CR plays a bigger role in reducing the total run time than the compression speed.

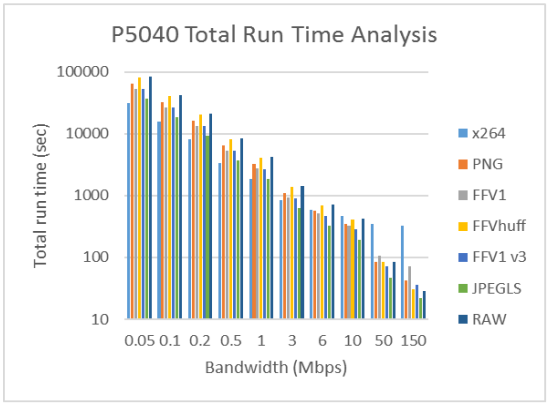
FFVhuff, which was the fastest encoder in terms of single-core execution time, actually performed the worst out of all of the encoders in terms of total run time. This reversal is because FFVhuff does not compress OPIR video efficiently, with an average CR of only 1.05 as compared to the BSS plus x264 CR of approximately 2.75 [1].

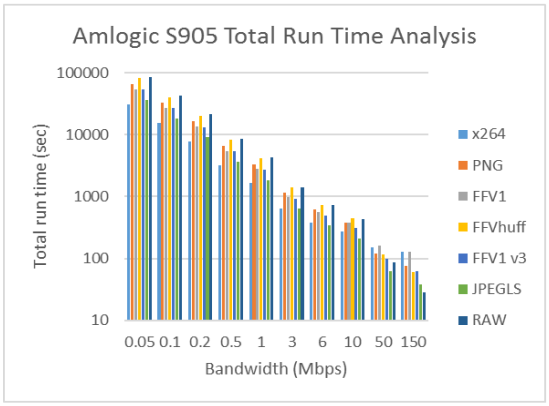
For the RAD5545 and the cases evaluated, compression speed attained was so limited that it is faster to downlink the raw data without compression, no matter the bandwidth. By contrast, on the other platforms, this scenario only occurred in some cases at high bandwidths.

6. CONCLUSIONS

The system designer must take into account many variables when deciding on the best solution to meet their requirements. They must not only consider bandwidth constraints but also the total run time it takes to execute the preprocessing technique, encoding, and transfer. If the total run time takes longer to execute than it does to downlink the raw data, then preprocessing and encoding may offer no advantage. This paper presented an analysis on four







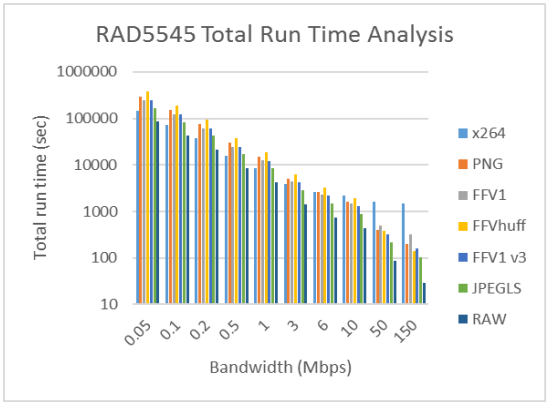


Figure 6: Total run time on (a) Zynq XC7Z020, (b) P5040, (c) S905, and (d) RAD5545

platforms where execution times of 16-bit encoders were compared with those of BSS plus x264 8-bit encoding. Since we did not have to perform BSS with 16-bit encode, execution times when BSS was performed were much slower. On the Zynq, BSS plus x264 is 5.54× slower than FFVhuff, 97.33× slower on the P5040 and RAD5545, and 3.18× slower on the Amlogic S905. Overall, the results show that the P5040 is the fastest platform for executing 16-bit encoders, the S905 quad-core Cortex-A53 also performs well, and FFVhuff is the fastest single-core encoder out of those tested.

Future work will include studies with lossy compression on these sample video files, using the same preprocessing methods and platforms presented in this paper, since only lossless compression was performed herein. Some of the encoders, such as JPEG-LS and FFV1, are inapplicable, since they are only lossless. A camera sensor with a frame rate close to the OPIR sensor will also be used to try to achieve real-time preprocessing and encoding.

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

[3] 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.

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

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

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

[7] 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.

[8] T. Lovelly, D. Bryan, K. Cheng, R. Kreynin, A. George, A. Gordon-Ross, and G. Mounce, "A Framework to Analyze Processor Architecture for Next-Generation On-Board Space Computing," Proc. of IEEE Aerospace Conference, Big Sky, MT, Mar. 1-8, 2014.

BIOGRAPHY

An Ho *is a M.S. student in ECE at the University of Florida. He 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 in the NSF CHREC Center at Florida.*

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 in the NSF CHREC Center at the University of Pittsburgh.*

Alan George *is Professor of ECE at the University of Florida, where he serves as 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. In January 2017, the lead site of CHREC, his students, and he will move to the University of Pittsburgh, where Dr. George will 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.*

Ann Gordon-Ross *received her B.S. and Ph.D. degrees in Computer Science and Engineering from the University of California, Riverside (USA) in 2000 and 2007, respectively. She is currently an Associate Professor of ECE at the University of Florida and is a faculty member in the NSF Center for High-Performance Reconfigurable Computing (CHREC). Her research interests include embedded systems, computer architecture, low-power design, reconfigurable computing, dynamic optimizations, hardware design, real-time systems, and multi-core platforms.*