Taming the Memory Demand Complexity of Adaptive Vision Algorithms

Majid Sabbagh, Hamed Tabkhi, Gunar Schirner

Taming the Memory Demand Complexity of Adaptive Vision Algorithms

Majid Sabbagh, Hamed Tabkhi, Gunar Schirner

Department of Electrical and Computer Engineering
Northeastern University
Boston, MA, USA
{msabbagh, tabkhi, schirner}@ece.neu.edu

Abstract With the demand for utilizing Adaptive Vision Algorithms (AVAs) in embedded devices, serious challenges have been introduced to vision architects. AVAs may produce huge model data traffic while continuously training the internal model of the stream. This traffic dwarfs the streaming data traffic (e.g. image frames), and consequently dominates bandwidth and power requirements posing great challenges to a low-power embedded implementation. In result, current approaches either ignore targeting AVAs, or are limited to low resolutions due to not handling the traffics separately. This paper proposes a systematic approach to tackle the architectural complexity of AVAs. The main focus of this paper is to manage the huge model data updating traffic of AVAs by proposing a shift from compressing streaming data to compressing the model data. The compression of model data results in significant reduction of memory accesses leading to a pronounced gain in power and performance. This paper also explores the effect of different class of compression algorithms (lossy and lossless) on both bandwidth reduction and result quality of AVAs. For the purpose of exploration this paper focuses on example of Mixture-of-Gaussians (MoG) background subtraction. The results demonstrate that a customized lossless algorithm can maintain the quality while reducing the bandwidth demand facilitating efficient embedded realization of AVAs. In our experiments we achieved the total bandwidth saving of about 69% by applying the Most Significant Bits Selection and BZIP as the first and second level model data compression schemes respectively, with only about 15% quality loss according to the Multi-Scale Structural Similarity (MS-SSIM) metric. The bandwidth saving would be increased to 75% by using a custom compressor.

1 Introduction

The demand for vision capabilities in embedded devices is rising more than ever. Embedded devices, ranging from tiny medical implants to smart cars and distributed smart cameras, need advanced vision capabilities and visual scenes analysis. Among different types of vision algorithms, the need is toward the algorithms that can dynamically adapt to the varying scene conditions. AVAs are considered as the dominating class of algorithms for advanced visual analysis. They are based on the machine-learning principles and are able to capture the runtime changes in the scene (e.g. MoG background subtraction and Support Vector Machine (SVM)).

While AVAs have been realized for fairly long time in algorithm-development environment (e.g. Matlab), their embedded low power realization is still very challenging.

Embedded devices are bounded in computation/communication resources with limited energy/power budget. In contrast, AVAs demand for a significant computation and communication capabilities which results in a power consumption far beyond the embedded system budgets. Realization of computation through custom design and High-Level Synthesis is well-formulated. On the other hand, communication appears as the primary bottleneck hindering implementation of AVAs on embedded devices.

The main limitation of AVAs is significant communication traffic imposed by algorithm itself. Due to inherent learning properties of AVAs, they maintain and continuously update model data of the scene. The model data is algorithm-intrinsic and its existence is independent of algorithm implementation. The size of model data is often very large exceeding the capacity of today's on-chip memories, forcing the designers to utilize off-chip memory. In result, accessing and updating the model data results in huge off-chip bandwidth demand and its associated power consumption. For instance, the bandwidth demand for updating the MoG background subtraction algorithm at Full-HD resolution is about 8GB/s, based on the analysis of the standard OpenCV algorithm. This contributes to about 90% of total power consumption [1]. Therefore, to open efficient realization of AVAs on embedded devices, the first step is to manage huge communication demands associated with the model data.

Existing approaches often ignore embedded realization of AVAs and focus on the non-adaptive ones, or only implement AVAs at very low resolutions (300*400) [2,3,4]. However, current trend is toward utilizing AVAs to deliver advanced vision capabilities at Full-HD resolution (1920*1080). Overall, optimizing the model data has received less attention despite being crucial for real-time low-power implementations of AVAs. The need is toward systematic approaches that can provide a guideline on how to efficiently manage model data communication traffic.

This paper introduces a system level approach for taming the memory demand complexity of Adaptive Vision Algorithms. The main goal of this research is to open a path toward efficient management of model data traffic. By focusing on model data, we explore the opportunity of shifting the current trend in compressing the streaming data (i.e. applying different video/image encoding methods), toward compressing the model data to reduce the bandwidth and power. Following a system-level approach, this paper also explores the effect of different classes of compression algorithms, lossy and lossless, on both bandwidth reduction and resulting quality. Based on the observations, this paper offers design choices and trade-offs for finding the best compression methods.

For the purpose of exploration, this paper focuses on the example of MoG background subtraction [5]. Our results on the example of MoG demonstrate 50% reduction in communication bandwidth by applying a lossy linear compression (Most Significant Bits selection) with minimal quality loss. A higher bandwidth saving 69% can be achieved by BZIP general-purpose lossless compression with no quality loss. Further bandwidth saving is also achieved by a customized compression scheme (e.g. 75% in [6]).

This paper is organized as following: Section 2 overviews relevant prior work. Section 3 briefly provides background and additional motivation. Following that, Section 4 describes our systematic approach for bandwidth quality trade-off on different class of compression algorithms. Section 5 concludes the paper and touches on future work.

2 Related Work

The embedded realization of vision algorithms is still at early stages. Most of the previous work on embedded vision have been bounded to basic vision filters, e.g., Canny edge detection, with regular computation and much less communication demand [7,8]. Only few researchers have targeted adaptive vision algorithms (e.g., MoG, KLT, optical flow) on embedded devices. What all previous approaches share in common is lack of insight about source and nature of the traffic. Therefore, they mainly propose a common communication interface for transferring all types of data. Thus, they either ignore the algorithm-intrinsic traffic, or assume that it is hidden in the communication hierarchy. In the result, their proposed solutions works at low resolutions (300*400) which is far below Full-HD resolution (1920*1080) [2,3,4,9]. However, in [1] authors propose Data Separation concept for Adaptive Algorithms, in which the streaming data is handled differently compare to the model data. That helped them to realize the MoG algorithm for background subtraction in Full-HD resolution at 30 frames per second. We based our studies on [1] for applying different compression schemes on model data and analyzing its effect on the system performance and output quality.

A recent work [6] hints about the significant communication volume of MoG background subtraction algorithm. It also provides a promising lossless compression method which can reduce the bandwidth demand for updating MoG parameters down to 50%. Although their compression method sounds to be very efficient and promising, their approach lacks a holistic approach which studies different compression schemes and can generalize the concept of compression on model data for AVAs.

The general observation is that optimizing the model data has received much less attention despite being crucial for real-time low-power implementation of AVAs. In contrast to previous approaches, this paper focuses on optimizing and managing huge communication demand for updating and accessing model data in AVAs.

3 Background

This section briefly provides background information on traffic separation in the context of AVAs, and for MoG background subtraction algorithm as an example of AVAs.

3.1 Data Separation

Data separation, proposed by [1], distinguishes two types of data: streaming data and model data. Fig. 1 highlights these two types of traffic in the context of AVAs. The streaming data (pixels) is the input/output data to the algorithm, while the model data is the intrinsic part of algorithm for realizing targeted processing. In AVAs, typically the model data is much bigger than streaming data. The size of model data exceeds the capacity of today's on-chip memories, forcing the designers to utilize the off-chip memory. Furthermore, since the model data should keep up with the streaming data, the bandwidth demand for reading and writing the model data from/to the memory would be a real limitation. Data Separation provides the opportunity for targeted optimization on both streaming data and model data.

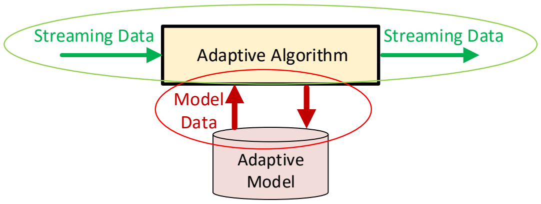


Figure 1: Streaming data vs model data

The traffic separation also streamlines the construction of complete vision flow out of multiple AVAs executing different parts of applications over the streaming data. Each AVA has its own model data which hits the memory hierarchy while streaming data is passed across the kernels. This further motivates us to have a systematic solution toward ever increasing communication complexity of AVAs. The Data Separation insight helps us to explore and tailor the compression opportunities on model data.

3.2 MoG Background Subtraction

MoG background subtraction is a very good representative of AVAs [5]. In this paper, we also use this algorithm to present our results. MoG is used in vision computing for identifying moving objects from the background scene. Fig. 2 includes the MoG coarse-grain mathematical formulation. MoG uses multiple Gaussian distributions, also called Gaussian components, to model a pixel's background. Each Gaussian component has its own set of Gaussian components: weight ω_i, an intensity mean μ_i and a standard deviation σ_i. In Full-HD resolution, for storing all Gaussian parameters about 74 MB of storage is required which may exceed the on-chip memory capacity available in embedded platforms. As a new pixel arrives, all Gaussian parameters are updated to track BG changes of the pixel at frame basis. On the other hand, the bandwidth demand for updating the Gaussian parameters, assuming 32 bit per Gaussian parameter with 3 Gaussian components per pixel, is about 8 GB/s (for processing at Full-HD resolution 60fps), which is 30 times more than streaming bandwidth, which is 265 MB/s for transferring 16-bit input pixel and 1-bit output foreground mask.

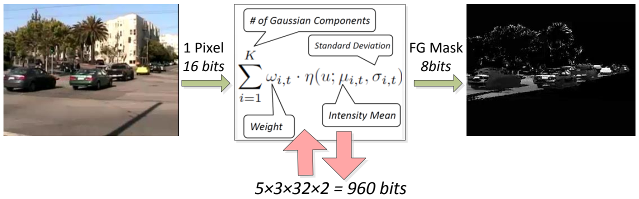


Figure 2: Memory Access per Pixel in MoG algorithm

4 Systematic Model Data Compression

4.1 System-Level Roadmap

Real-time embedded realization of AVAs forces the system architects toward systematic approaches. Fig. 3 highlights our design flow for tackling the complexities of AVAs. It

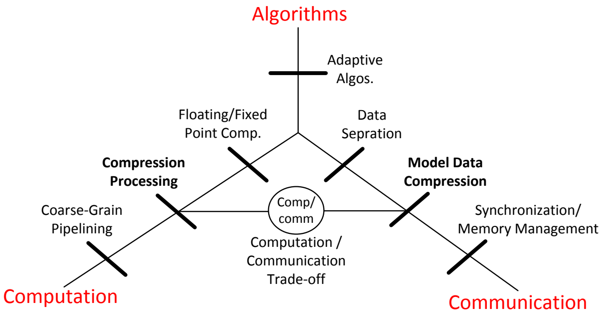


Figure 3: System-Level design flow

starts from analysis of algorithms to identify orthogonal axises of optimization with separated axises for computation and communication. There is a computation/communication trade-off at which point the computation and communication axises meet. The trade-off occurs when compressing the model data adds to the computation demand while reduces the bandwidth demand.

In this study, we mainly focus on communication axis because of significant communication demand in AVAs for periodically accessing and updating the model data. Complexities in computation axis, include but not limited to, multi-dimensional processing, complex operations, floating point computation which can be explored separately and are not part of this study.

Using the data Separation insight, we propose a shift from current trend in compressing streaming data toward compressing model data. As shown in Fig. 4, compression/decompression units could be placed in the access interface of model data, providing the opportunity of significantly reducing the model update bandwidth.

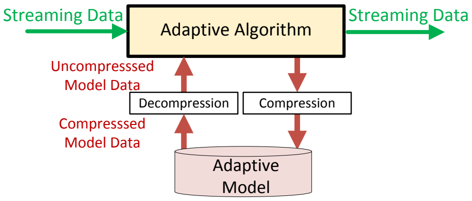


Figure 4: Compression/Decompression on model data access

There are two major categories in compression methods, lossy and lossless. In lossy compression, there is a trade-off between the achievable bandwidth saving and output quality while in the lossless compression schemes there would be no quality loss probably with the cost of higher computation demand. Overall, the primary metrics needed to be considered for choosing a suitable compression algorithm are: (1) achievable bandwidth saving, (2) quality effect (lossy/lossless), and (3) computation demand for compressing/de-compressing model data.

4.2 Experimental Setup

For the purpose of study, we focus on MoG background subtraction algorithm explained in Section 3.2. We modeled MoG in SpecC System Level Design Language (SLDL)

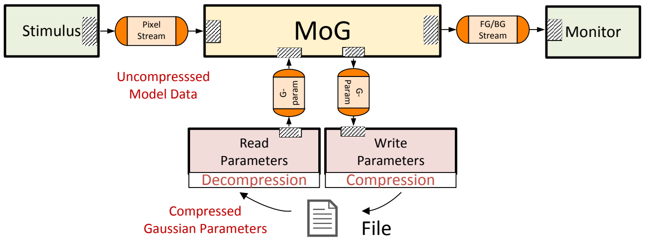


Figure 5: High-level diagram of evaluation model for MoG

[10]. The high-level diagram of the experimental model is shown in Fig. 5. The stream of input pixels will be fed from stimulus block to the MoG which at the same time receives the corresponding parameters, i.e. model data, from the Read Param block. Read Param block receives the decompressed parameters from the decompression unit. Decompression unit reads the compressed model data from a file, decompress it, and sends the individual parameters out when the corresponding pixel arrives at input of MoG. Then, after finishing the processing, MoG block outputs foreground masks as the output stream and updated parameters as model data. Model data will be written back to a file after being compressed by the compression unit. In the following section, we will study the effect of applying different compression methods on MoG's model data.

During the exploration, we assess the quality against the ground-truth (MoG with no compressed Gaussian parameters). We use MS-SSIM metric. MS-SSIM focuses on the structural similarity between two frames which is more similar to human perception [11]. MS-SSIM quantifies the quality as a value between 0 to 1 where a higher value means closer similarity and thus lower quality loss compared to ground-truth.

4.3 Evaluation of Compression Schemes

MSBSel The simplest compression method is to statically select the most significant bits of parameters. We call this method Most Significant Bits Selection (MSBSel). Although MSBSel is a lossy compression, it has almost no computation overhead especially considering a hardware implementation. We call the model with MSBSel compression method the reference model in later explanations.

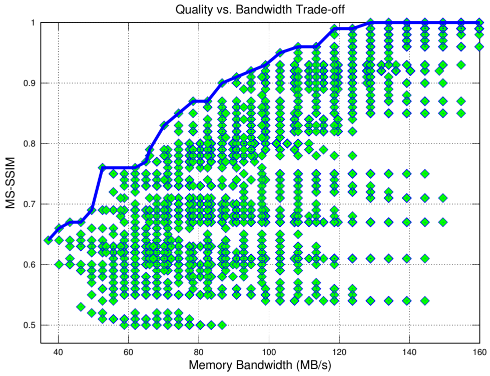


Figure 6: Quality-Bandwidth trade-off in MSBSel compression scheme

This compression scheme introduces the Quality-Bandwidth trade-off as illustrated in Fig. 6. Based on Fig. 6, the quality and bandwidth have a non-linear relationship and it is possible to reduce bandwidth without quality loss (in the bandwidth range from 160 MB/s down to about 120 MB/s). Moreover, by reducing the bandwidth down to 50%, i.e. 80 MB/s, quality will not drop more than 15% according to MS-SSIM quality metric. Although this is due to the nature of MoG algorithm, but same trend could be observed for other AVAs. Selecting fewer bits leads to higher bandwidth saving but also higher degradation in quality. A high dynamic range or precision is essential for accurate background subtraction. This raises a need for complex/advanced compression algorithms to further reduce the bandwidth requirements for updating model data.

JPEG-2000 To further explore the compression effect on MoG parameters, JPEG-2000 compression applied as the second-level compression after MSBSel to evaluate if it can reduce the bandwidth without affecting the quality significantly. JPEG is a well-known image compression method developed by Joint Photographic Experts Group [12]. JPEG-2000 is the newer version of JPEG, which also supports 16-bit gray-scale images.

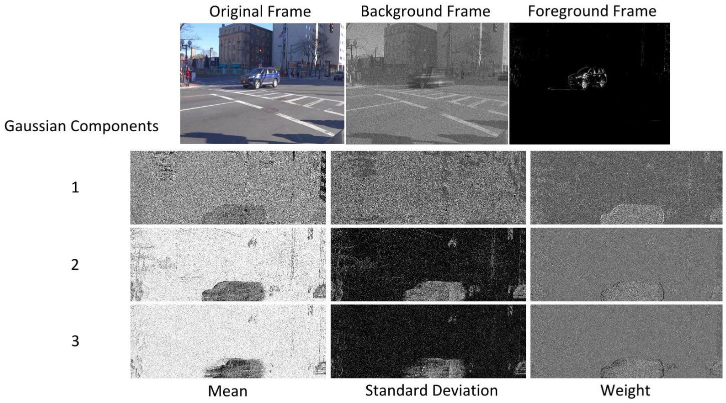
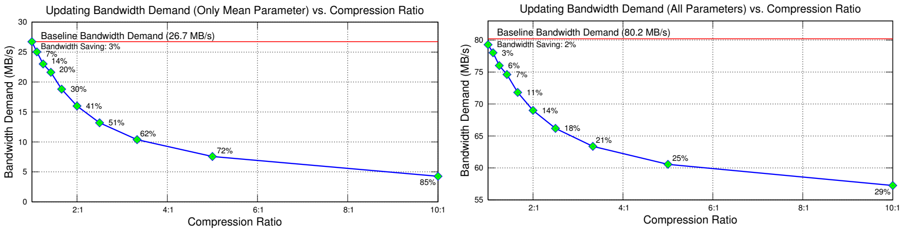


Figure 7: Visual structure of Gaussian parameters.

Since JPEG has been customized for compressing images, we only limit the JPEG compression for parameter *Mean*. Parameter *Mean* keeps background mean values for pixels and it has a visual structure similar to an image. Fig. 7 illustrates the visual structure of all Gaussian parameters. On top from left to right, the original, background and foreground frames are shown. Below the frames, Fig. 7 also presents the images of all Gaussian parameters for three Gaussian components: *Mean*, *Standard Deviation* and *Weight*, from left to right.

Fig. 8a shows the effect of JPEG-2000 compression on *Mean* parameter over the bandwidth demand for updating parameter *Mean*. Fig. 8b plots the total bandwidth saving (over all parameters), by compressing only parameter *Mean*. For this experiment, a sequence of 650 frames with resolution of 320x240 has been chosen as the input test set. Both figures show the effect of different JPEG compression ratios (2:1 – 10:1). The JPEG-2000 is significantly reducing the bandwidth demand for updating the *Mean* parameter. For the best compression ratio of 0.1 (10:1), the bandwidth saving for updating parameter *Mean* is about 85%, indicating that the size of compressed *Mean* image is on



(a) Only *Mean* parameter

(b) All Gaussian parameters

Figure 8: Bandwidth demand vs. compression ratio for JPEG-2000 compression

average 15% of its original size. Also, as shown in Fig. 8b, overall bandwidth saving is about 29%. The overall bandwidth saving is limited as JPEG-2000 is not applicable on other MoG parameters (Standard Deviation and Weight).

The bandwidth saving corresponding to the compression ratio was less than the expected value. For example, the expected bandwidth saving of 10:1 compression ratio, is 90%. However, in practice it is about 85%. The effect of noise on degrading the performance of compression methods is already studied in literature. For example, a study in [13] shows that noise reduces the inter-pixel correlation which compression ratio increased. Some approaches, such as [14] and [15], filters the effect of noise from the source image. However, all based on assumption that feature of filtering solutions is that they consider noise in images as unwanted data. In contrast, the visual distortion that is observable in MoG parameters are actually the values produced by the algorithm and are required in next iteration for computation. Therefore, eliminating the distortion which is seen in MoG parameters in fact reduce the quality of MoG.

To study the effect of JPEG bandwidth saving on the MoG quality (the foreground masks), we compared the output of MOG with JPEG compression against the reference model. The outputs are compared using the MS-SSIM quality metric in a simple scene, with few object movements and variations, and a complex scene, with lots of object movements and crossings. Fig. 9 plots the quality across total bandwidth saving when applying JPEG compression for parameter *Mean*. The maximum achievable output quality is about 0.46 out of 1, in the simple frame sequence. This is even worse in complex frame sequence which the maximum output quality is 0.22 with bandwidth saving of only 2% on average. With increasing compression rate to further reduce the bandwidth, the quality degradation would be more pronounced reaching to a point which basically MoG is not functional anymore.

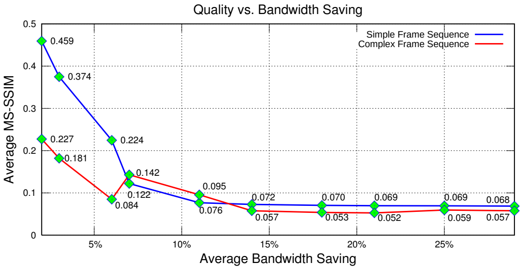


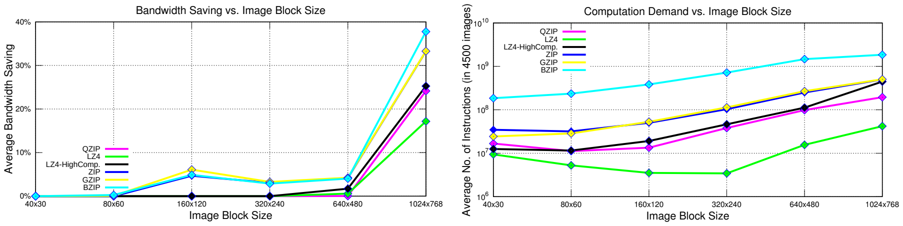
Figure 9: Quality-Bandwidth trade-off when for simple and complex frame sequences

Overall, we conclude that JPEG-2000 have severe degradation effect on the final output quality of MoG. We can trace back this issue to the adaptive behavior of the MoG algorithm, meaning that since the MoG parameters are recursively accessed and updated, even a small error can accumulate, resulting in degradation of background subtraction robustness and eventually output quality loss. All of these observations motivate us to explore other possibilities for compression of model data, therefore we will look into the opportunity of using lossless compression schemes.

Lossless Compression The principal feature of lossless compression algorithms is that they do not affect the quality of data being compressed. Therefore, rather than quality loss, the designer may change the focus to other characteristics of compression algorithms, such as the achievable compression ratio and computation demand. In our study, five different lossless compression methods have been explored: QZIP, LZ4, BZIP, GZIP and ZIP. QZIP and LZ4 are two compression algorithms which are in high-speed categories, while BZIP, GZIP and ZIP are three regular compression methods. The experiments are done over 4500 parameter images that are the MoG parameters of 3 Gaussian components, each having 3 parameters (Mean, Standard Deviation and Weight) for 500 frames of original 1024x768 resolution. All of the lossless compression methods are applied as the second level compression after MSBSel. To further explore the granularity of source data which a certain algorithm will operate on, we divide the input frames to smaller blocks. A 1024x768 frame could be divided into 25.6 blocks of 40x30, 12.8 blocks of 80x60, 6.4 blocks of 160x120, 3.2 blocks of 320x240, 1.6 blocks of 640x480 and 1 block of 1024x768 resolution.

Fig. 10a presents the average bandwidth saving over different block sizes for all compression algorithms. The maximum bandwidth saving could be achieved, when using the 1024x768 block size. In fact, by reducing the block size to less than 1024x768, all algorithms perform poorly in bandwidth reduction, at the best case reaching to less than 10% bandwidth saving. The best average bandwidth savings captured in lossless experiments is about 38% percent, corresponding to BZIP compression scheme with block size of 1024x768.

Fig. 10b shows the average computation demand over different block sizes for all compression algorithms. To estimate the computation demand, we used Pin - Dynamic Binary Instrumentation Tool [16], from Intel. The block size of 1024x768, equal to full image size, leads to largest computation demand. Overall, the computation demand varies across the algorithms. In QZIP and ZIP, the lowest computation demand is achieved by using block size of 80x60, for LZ4 320x240 and for GZIP and BZIP 40x30. Fig. 10a and



(a) Bandwidth saving

(b) Computation demand

Figure 10: Bandwidth saving and computation demand over different block sizes.

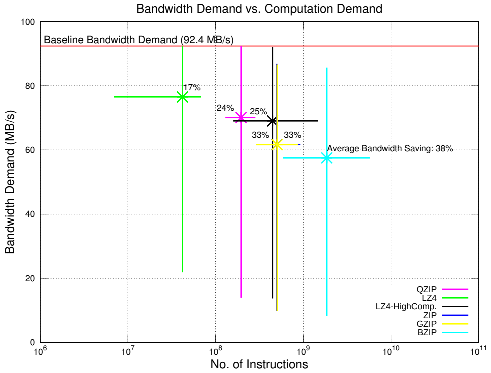


Figure 11: Bandwidth saving / computation demand trade-off

Fig. 10b also present the result of LZ4 algorithm in the high-compression mode (LZ4 High-Comp.). In high-compression mode, LZ4 could provide better compression ratio with the cost of higher computation demand.

Combining Fig. 10a and Fig. 10b, we can derive Fig. 11 showing the trade-off between bandwidth saving and computation demand. In Fig. 11, the numbers over the stars show the average bandwidth saving, while the crosses show the range of achievable bandwidth demand corresponding to a certain computation demand. For different algorithms, the trend is that lower bandwidth demand or higher bandwidth saving, is achievable by having more computation. Among all algorithms, LZ4 in fast-mode has the lowest computation demand, which is about 100 millions instructions for compression and decompression of a frame, and at the same time the lowest bandwidth saving of 17%, while the BZIP achieves highest bandwidth saving of 38% with the highest computation demand of about 1 billion instructions. Note that, computation demand of LZ4 for compression and decompression of a frame in high-compression mode is about 10 times higher than its computation demand in fast-mode.

Custom Compression The previous studied lossless compression methods are general algorithms developed to compress any sort of data. The results of our experiments illustrate that although there are lots of options for compressing the model data, but for AVAs such as MoG, which have special type of model data produced by statistical and non-linear processing, general purpose image or data compression methods might not provide the desirable performance for designers. There is a demand for lossless compression algorithm customized for compressing model data in context of AVA. This fact, motivates designers such as in [6] to devise custom model data compression schemes only for a certain type of algorithm. One preliminary example, has been already proposed by [6].

In [6], the authors propose a DPCM-based compression algorithm for compression of MoG parameters. Their algorithm, extracts and uses the inter-correlation of parameters for different Gaussian Components of MoG and intra-correlation of parameters for within the Gaussian Components, to represent every two parameters with one compressed parameter, in a lossless process, reducing the bandwidth demand down to 50%. In terms of computation demand, the FPGA implementation of their algorithm shows reasonable resource utilization of about 2282 LUTs and 1876 FFs in Virtex-5 FPGA, without any use of DSP Blocks and more importantly Block RAMs.

4.4 Results Summary

To summarize, Using the Data Separation insight, we explored the opportunity of applying different compression schemes on model data, to shift the compression trend from streaming data to model data. During the experiments we evaluated several trade-offs:

- **Lossy Compression**
    - Quality-Bandwidth Trade-off (for MSBSel only and MSBSel+JPEG-2000): For having better quality, higher bandwidth is needed
    - Bandwidth-Compression Ratio (for MSBSel+JPEG-2000): By changing the configurable compression ratio in JPEG-2000, different bandwidth demands are achievable
- **Lossless Compression**
    - Computation Demand-Block Size Trade-off (For all 5 lossless algorithms): Different block sizes lead to different computation demands, highest computation demand is when the block size is equal to the whole frame being compressed
    - Bandwidth Saving-Block Size Trade-off (For all 5 lossless algorithms): Different block sizes lead to different Bandwidth Saving, biggest saving in Bandwidth achieved when largest block size is used
    - Bandwidth Demand-Computation Demand Trade-off (For all 5 lossless algorithms): For having lower bandwidth demand or highest bandwidth saving, there is need for more computation
    - Bandwidth Demand-Computation Demand Trade-off for LZ4 algorithm in high-compression mode and fast-mode: By using LZ4 in high-compression mode, higher bandwidth saving is achievable with the cost of higher computation demand

Overall, by applying BZIP as the second level compression scheme over MSBSel, the total bandwidth saving of about 69% is achievable, while the bandwidth saving could be increased to about 75% by using a custom compressor instead of BZIP. The overall output quality loss would not be more than 15% according to the MS-SSIM metric for both of these cases. These trade-offs could help the system architects for AVAs, choose the right compression algorithm for model data based on the application requirements. Furthermore, designers might have to use or devise tailored algorithms for AVAs' model data, such as the compression method proposed in [6] for compressing the MoG parameters.

Altogether, using the Data Separation insight and by applying compression on model data, realization of AVAs could be facilitated as the overall system bandwidth demand became manageable, taming the complexity of AVAs realization. We would consider power consumption in future works as it is a very important factor in embedded systems, while in this work we focus on evaluating the compression ratio and computation demand in lossless compression schemes.

5 Conclusions

This paper proposes a systematic approach for tackling the complexity of AVAs. We focus on the main challenge which is the communication complexity due to huge bandwidthdemand for updating model data in AVAs. Using the Data Separation insight and by applying compression on model data, realization of multiple vision kernels could be facilitated on a single platform. Also, throughout our study we evaluated different lossy and lossless compression algorithms, providing the system architects with various trade-offs and intuitions for choosing the right method of compression for model data.

References

1. H. Tabkhi, M. Sabbagh, and G. Schirner, “Power-efficient real-time solution for adaptive vision algorithms,” *IET Computers & Digital Techniques*, pp. 16–26(10), 2015.
2. J. Xu et al, “A case study in networks-on-chip design for embedded video,” in *Design, Automation and Test in Europe (DATE)*, vol. 2, 2004, pp. 770–775 Vol.2.
3. T. Lv et al, “A methodology for architectural design of multimedia multiprocessor socs,” *IEEE Design Test of Computers*, vol. 22, no. 1, pp. 18–26, 2005.
4. G. Chen et al, “Energy savings through compression in embedded java environments,” in *International Symposium on Hardware/Software Codesign*, ser. CODES, 2002.
5. C. Stauffer and W. E. L. Grimson, “Adaptive background mixture models for real-time tracking,” in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2, 1999, pp. –252 Vol. 2.
6. K. Ratnayake and A. Amer, “Embedded architecture for noise-adaptive video object detection using parameter-compressed background modeling,” *Journal of Real-Time Image Processing*, 2014.
7. Xilinx, “Programming vision applications on zynq using opencv and high-level synthesis,” *Xilinx technical report*, 2013.
8. Z. Tang and D. Shen, “Canny edge detection codec using vlib on davinci series dsp,” in *Computer Science Service System (CSSS), 2012 International Conference on*, 2012.
9. K. Swaminathan, G. Lakshminarayanan, and S.-B. Ko, “High speed generic network interface for network on chip using ping pong buffers,” in *International Symposium on Electronic System Design (ISED)*, 2012, pp. 72–76.
10. A. Gerstlauer, R. Dömer, J. Peng, and D. D. Gajski, *System Design: A Practical Guide with SpecC*. Kluwer Academic Publisher, 2001.
11. Z. Wang and et al., “Image quality assessment: from error visibility to structural similarity,” *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.
12. JPEG. [Online]. Available: http://www.jpeg.org
13. S.-C. Lo, B. Krasner, and S. Mun, “Noise impact on error-free image compression,” *Medical Imaging, IEEE Transactions on*, 1990.
14. P. Cosman, R. Gray, and R. Olshen, “Evaluating quality of compressed medical images: Snr, subjective rating, and diagnostic accuracy,” *Proceedings of the IEEE*, 1994.
15. P. W. Melnychuck, M. J. Barry, and M. S. Mathieu, “Effect of noise and mtf on the compressibility of high-resolution color images,” 1990.
16. “Pin - a dynamic binary instrumentation tool,” https://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool, accessed: 2015-08-30.