Algorithm and Architecture Co-Design of Mixture of Gaussian (MoG) Background Subtraction for Embedded Vision

Hamed Tabkhi
Departement of Electrical and Computer Engineering
Northeastern University
Boston (MA), USA
Email: tabkhi@ece.neu.edu

Robert Bushey
Embedded Systems Products and Technology
Analog Devices Inc. (ADI)
Norwood (MA), USA
Email: robert.bushey@analog.com

Gunar Schirner
Departement of Electrical and Computer Engineering
Northeastern University
Boston (MA), USA
Email: schirner@ece.neu.edu

***Abstract*—Embedded vision is a rapidly growing and challenging market that demands high computation with low power consumption. Carefully crafted heterogeneous platforms have the possibility to deliver the required computation within the power budget. However, to achieve efficient realizations, vision algorithms and architectures have to be developed and tuned in conjunction.**

This article describes the algorithm / architecture co-design opportunities of a Mixture of Gaussian (MoG) implementation for realizing background subtraction. Particularly challenging is the memory bandwidth required for storing the background model (Gaussian parameters). Through joint algorithm tuning and system-level exploration, we develop a compression of Gaussian parameters which allows navigating the bandwidth/quality trade-off. We identify an efficient solution point in which the compression reduces the required memory bandwidth by 63% with limited loss in quality. Subsequently, we propose a HW-based architecture for MoG that includes sufficient flexibility to adjust to scene complexity.

I. Introduction

With the significantly growing processing capabilities, vision algorithms have started to migrated toward embedded platforms. Embedded vision computing refers to deploying visual analysis and computer vision capabilities to embedded systems [1], covering a variety of markets/applications with a high diversity in algorithms and conflicting requirements [1], [2]. Few examples of fast growing markets that involve high-performance vision computing are Advanced Driver Assistance System (ADAS), industrial vision and video surveillance. The surveillance market alone is estimated to more than triple from $11.5 billion in 2008 to $37.5 billion in 2015 [3]. The global value of ADAS market has an even stronger projected growth: 13-fold from $10 billion in 2011 to $130 billion in 2016 [4]. All these markets share the same top-level functionality of real-time video processing, including modeling of captured scenes and object analysis.

A significant challenge in embedded vision computing is to define and design an efficient architecture that satisfies requirements across multiple markets (cost, performance power). A large portion of vision applications is highly compute intense as they are set to work on High-Definition (HD) resolution under real-time constrains. At the same time, vision processors are very cost sensitive with low power budget as they need to be used in a very large volume at embedded environments. For HD resolutions, vision algorithms require 10, 20 or even more billions of operations per second. In result, the challenge is to architect solutions that can burn very little power (sub watt in many cases), while achieving high performance.

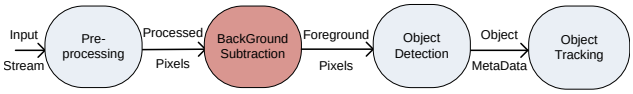


Figure 1: Example vision flow.

Fig. 1 conceptualizes a coarse-gain vision flow for a video surveillance or a patient monitoring solution. A typical flow starts with *Pre-processing* that reduces noise, adjusts contrast (and other parameters), and converts RGB to gray. Then, *Background Subtraction* separates the foreground (FG) - which are moving objects - from the background (BG). Based on the FG, *Object Detection* identifies objects which are then tracked by *Object tracking* across multiple frames. Object tracking is then the basis for generating events. For the surveillance market, analysis examples may include identification of an object left behind and detecting when a secured zone is entered. All five application blocks are common in the area of embedded vision computing with more emphasis in video surveillance market.

A common block across many vision applications is background subtraction and can also be considered as a primary kernel. In a nutshell, background subtractions separates foreground pixels from a static background scene [5]; generating foreground mask. Background subtraction is challenging as it operates in real-time on every pixel of the input stream. Example applications utilizing background subtraction include video surveillance, industry vision, and patient monitoring systems.

In this article, we specifically concentrate on *Background Subtraction* as a very challenging and computationally intense algorithm. For scenes with a static camera position, Mixture of Gaussian (MoG) [6][7] is frequently used. Comparing against other algorithm candidates, MoG is an efficient algorithm in terms of computation and memory demand and yields good quality. Furthermore, being an adaptive algorithm MoG offers many algorithm / architecture co-design opportunities.

For algorithm / architecture co-design of MoG, we capture it in the SpecC [8] SLDL. Our early profiling indicates that a software realization of MoG is infeasible needing up to 20 DSP cores even under optimistic assumptions. Consequently, we propose a hardware-centric solution with some degree of flexibility to adjust the quality according to the scene complexity. As one aspect of algorithm / architecture co-design we propose a parameter compression which poses a tradeoff between image quality and external memory bandwidth. We analyze the trade-off driven by the compression factor and arrive at a solution that shows a 63% reduction in memory bandwidth with acceptable loss in quality. The compression minimally increases the computation computation, but allows for a lower power and more cost effective solution. The resulting specification model serves as an input for a top-down downstream ESL flow for further fine grained exploration, synthesis and implementation.

The remainder of this paper is organized as follows. Section II overviews the MoG algorithm. Section III presents MoG's computation and communication demands. Section IV explores the bandwidth / quality trade-off. Section V conceptually introduces our proposed HW architecture. Finally, Section VI concludes the paper and touches on future work.

II. Mixture of Gaussians (MoG)

The survey in [9] categorizes algorithm candidates for *Background Subtraction*. They range from history-based implementations [10][11] to adaptive learning algorithms [12][13]. History-based algorithms operate on a history of the past N video frames leading to high memory capacity requirements for storing the frames and well as large memory bandwidth requirement for repeatedly accessing the frame history. Conversely, adaptive algorithms track the background with a model. They operate only on the current video frame to iteratively update the model. Adaptive algorithms typically handle gradual variations better and have lower memory requirements.

We hence focus on adaptive algorithms and selected Mixture of Gaussians (MoG) [5]. MoG provides sufficient quality with acceptable memory and computation requirements. MoG core equation (1) uses K Gaussian components, each with a weight $\omega_{i,t}$, an intensity mean $\mu_{i,t}$ and a standard deviation $\sigma_{i,t}$:

$$\sum_{i=1}^{K} \omega_{i,t} \cdot \eta(u; \mu_{i,t}, \sigma_{i,t}) \qquad (1)$$

MoG uses multiple Gaussian distributions, also called Gaussian components, to model a pixel's background. Each pixel has an own set of Gaussian components. Each Gaussian component is captured in three parameters: *Mean* (BG gray scale intensity), *Weight* (as indicator of Gaussian component strength) and *Deviation* (distribution over *Mean*).

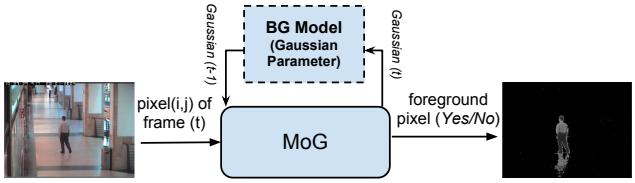


Figure 2: Adaptive background subtraction by MoG.

With a new pixel coming in, the Gaussian parameters are updated at frame basis to track BG changes. Fig. 2 illustrates a coarse-grained flow of the MoG. Each pixel's background value is modeled by three to five Gaussian components (distribution). The pixel's Gaussian components are updated, based on learning factor, taking into account a new incoming pixel value. If none of the Gaussians sufficiently matches the new pixel value, that pixel is declared as a foreground. MoG offers configurability [14] (e.g. # of components) to adjust to scene complexity or desired output quality.

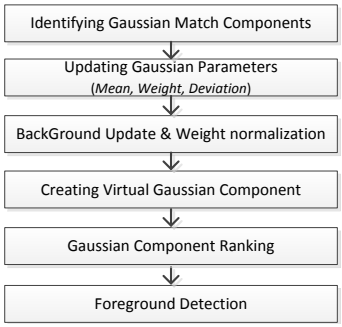


Figure 3: MoG algorithm overview.

Fig. 3 outlines an MoG algorithm flow chart. Each pixel's Gausian components are updated individually (independent of other pixels). MoG first classifies the pixel's Gaussian components into *match* or *non-match* components. A component matches, if the component's *mean* is within a predefined range of the current pixel value. Gaussian parameters are updated based on *match* classification. In case that no match exists, algorithm creates a new Gaussian component, called virtual component, and replaces the virtual component with the Gaussian component with smallest *weight*. Then, the components are ranked based on their *weight* over *standard deviation* ratio. Starting from the highest rank component, the algorithm declares a pixel to be background based on rank and closeness in matching with current value, if there is no match component, the pixel is identified as foreground.

One advantage of MoG is its systematic configurability. By changing the number of Gaussian components, the quality can be adjusted at cost of additional computation and memory bandwidth. In the following sections, we describe the resource demands of MoG and its realization in hardware.

III. MoG Demands

Understanding the computation and communication demands of an application are important, as they restrict the possible realization space (e.g. SW, custom HW). To profile MoG, we captured *Mixture of Gaussian (MoG)* algorithm in the SpecC System level Desing Language (SLDL) [8]. We used the source-level profiling of System-on-Chip Environment (SCE) [15] to determine computation and communication demands. The following subsections illustrate the computation and communication demands individually.

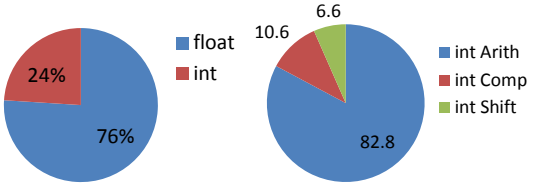
A. Computation Demands

Table I presents the MoG computation demands in number of operations per second [GOPs/Sec]. Profiling is based on Full-HD (1920x1080) and half-HD (1280x960) resolution, each with a 60Hz frame rate. *Mixture of Gaussians (MoG)* is very demanding in computation with around 24 GOPs/Sec for Full-HD. Table I roughly targets the computation demand to resource utilization using very optimistic assumptions – basically giving very optimistic lower bound of needed resources. Assuming execution on a Blackfin 52x core with 600MHz [16], 20 cores and 13 cores are respectively required for executing MoG in Full-HD and Half-HD resolutions respectively.

Table I: Resource utilization of MoG at 60Hz frame rate.

Image size	Computation Demand [GOPs/Sec.]	Blackfin cores Float type [#]	# Blackfin cores Int. type [#]
1920*1080	24.3	20	13
1280*960	14.4	12	8

One possible solution for reducing the MoG computation demands is replacing floating-point operations with integer computation. Our computation profiling, highlighted in Fig. 4a, shows that approximately 76% of ALU operations (and 60% of overall operations) of MoG are floating-point operations. To improve the resource utilization, we have translated our MoG model in SpecC to a 32-bit integer implementation. After translation, 100% of ALU operations are of integer type. Fig. 4b overviews the contribution of different integer ALU operations. More than 82% of ALU operations is integer arithmetic (+,-,*,/). The fixed-point conversion speeds up only by 1.5 time, as only 60% of the total computation was affected. Still, 12 and 8 Blackfin DSPs are required for fixed-point implementation in software.



(a) Floating-point type (b) Integer type

Figure 4: Operation Types of MoG kernel

MOG's computational complexity make it inefficient and too costly to implement in software and conversely an excellent candidate for a hardware realization. However, the challenge arises to maintain enough flexibility in the custom-HW implementation to support all targeted markets. Before progressing into the custom-HW design, the communication demands need to be evaluated to guide the system-level design.

B. Communication Demands

Table II summarizes communication demands for MoG memory accesses in number of bytes transferred per second [GBs/Sec] and also includes example memory interface utilization. Table II reveals the very high communication demand for MoG. For each pixel, the background model (consisting of 3-5 Gaussians with 3 parameters each) is updated with each incoming image frame. For each frame, depending on resolution, 248MB and 146MB are accessed respectively for full-HD and half-HD resolutions when reading and writing Gaussian parameters. This results in 7464MB (4360MB) of traffic per second.

Table II: Estimated resource utilization of Mixture of Gaussians (MOG) algorithm with the frame rate of 60 Hz.

Image size	Bandwidth [MB/Sec.]	Utilization DDR3 1066 [%]	# Utilization LPDDR2 800 [%]
1920*1080	7440	88	Saturated
1280*960	4380	48	70

The high communication demand stems from fetch and write back of Gaussian parameters (*mean*, *weight* and *standard deviation*). Assuming 32bit parameter length per Gaussian parameters and 3-5 Gaussian components per pixels, 72 to 120 Byte memory access per pixel is required for updating Gaussian parameters. This very high communication demands prompt us to further investigate the MoG algorithm. One possible approach is tuning of the Gaussian parameters to reduce the memory bandwidth requirements with minimal lost in output quality.

IV. Bandwidth / Quality Tradeoff

Assuming a HW implementation of MoG, the computation demand is not problematic as sufficient parallelism (across Gaussian components) and pipelining opportunities (within components) exist. Our main goal for tuning is to reduce the memory bandwidth. We therefore explore the bandwidth / quality trade-off to identify the lowest possible bandwidth with acceptable quality. Bandwidth reduction is possible by reducing the number of Gaussian components or reducing the bit-width of parameters when stored in memory.

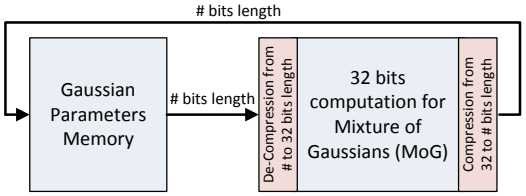


Figure 5: compression/Decompression of Gaussian Parameters.

To reduce the memory bandwidth we propose adding a support block to compresses / de-compress parameters interfacing with memory as shown in Fig. 5. The additional module should be configurable to compress three Gaussian parameters: *mean*, *deviation* and *weight* at different bit width. While internal processing is still performed at 32 bits, parameter values are compressed to a fixed length between 8 and 24bits each. Upon read, the compressed values are transformed back to 32 bit for processing. For simplicity at this stage, we compress through discretization focusing on the most significant bits. To analyze the tradeoff, we exhaustively explore across lengths of Gaussian parameters (8 to 24bits) as well as number of Gaussian components. The following subsections describe our bandwidth / quality exploration in detail.

A. Exploration Setup

Driven by the high demand on memory bandwidth, our first goal is to reduce this resource demand. Bandwidth can be reduced by using fewer Gaussian components or by reducing number of bits for each parameter when stored in memory. Each, however, will reduce the quality of background elimination depending on scene complexity. For quality assessment, we use MS-SSIM [17] as a good indicator for visual quality focusing on structural similarity rather than direct pixel comparison approach. We evaluate quality by comparing against a ground truth obtained from the reference algorithm (32-bit fixed point operation and 32-bit storage for each Gaussian parameters). To explore the tradeoff, we expanded our SpecC model to support parameters compression / de-compression at different bit width as well as configurable number of Gaussian components.

Table III: Scenes under the test

		
Simple scene (*Shopping Mall*)	Medium scene (*Subway Station*)	Complex scene (*SanFrancisco Street*)

Table IV: Estimated resource utilization of Mixture of Gaussians (MOG) algorithm with the frame rate of 30 Hz.

data set	BG Changes	FG Changes	Variation Variation	Object Object
Simple	no	no	no	no
Medium	no	yes	no	yes
Complex	yes	yes	yes	no

We investigate three scenes: *simple* an indoor shopping center scene with limited motion, *medium* a subway station and *complex* a busy outdoor scene of a street corner. Table III shows a sample frame of each scene. We characterize scene complexity based on the number of moving objects, illumination, shadows, and many other details. Table IV summarizes the features of each scene. The most challenging scene is San Francisco

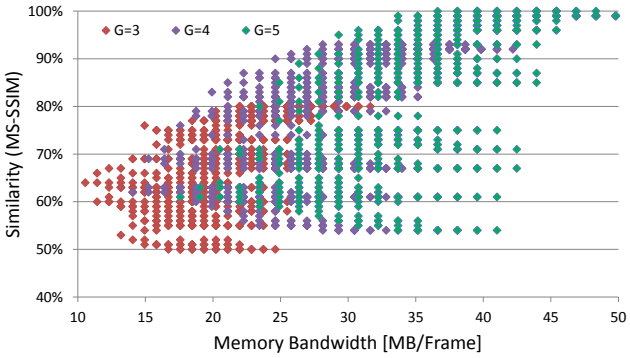


Figure 6: Quality / bandwidth exploration.

Street with variations from multi-modal changes in background to different kind of moving object at foreground as well as frequent changes in illumination.

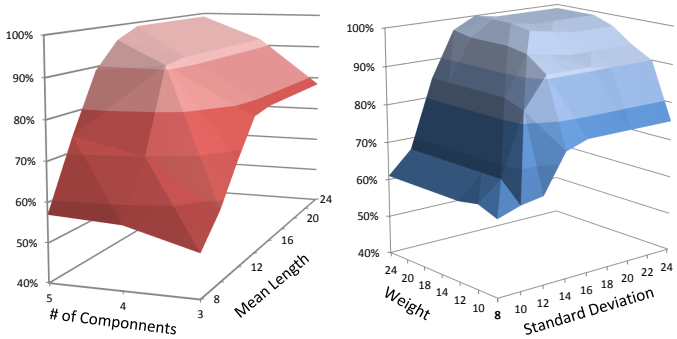
B. Parameters exploration

We systematically analyze and explore the impact of parameter bit-width (individually for *mean*, *weight*, and *deviation*) as well as number of Gaussian to the quality of background subtraction for all scenes. In this subsection, we only presents part of the exploration focusing on the complex scene.

Fig. 6 plots the trade-off for the *complex* scene with memory bandwidth on the x-axis and quality (MS-SSIM) on y-axis for all configurations (# Gaussians, parameter length). By increasing the number of Gaussian components quality improves at cost of bandwidth. However, quality between different number of Gaussians overlap. Furthermore, different configurations (e.g. in parameter length) yield the same bandwidth, however vary in quality. To identify which configurations yield best quality, we analyze individual dimensions (# Gaussians, parameter length).

Fig. 7a and Fig. 7b highlight two exploration examples for the complex scene. Fig. 7a shows the quality correlation between # of Gaussian components and length of *Deviation*. The other parameters (*Mean* and *Weight*) are kept at maximum length (32-bit).

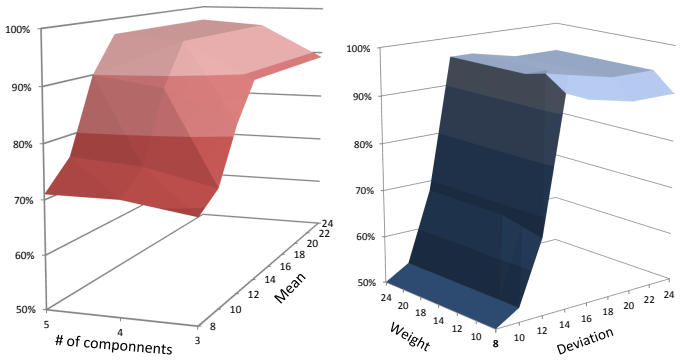
Increasing # of Gaussian components from 3 to 5 linearly improves quality due to the high scene complexity. This confirms that 5 Gaussian components are required in the complex scene.



(a) # of Gaussian components over Mean [bits]

(b) Weight [bits] over Standard Deviation [bits]

Figure 7: Exploration for complex scene



(a) # of Gaussian components over Mean [bits] (b) Weight [bits] over Standard Deviation [bits]

Figure 8: Exploration for the medium scene

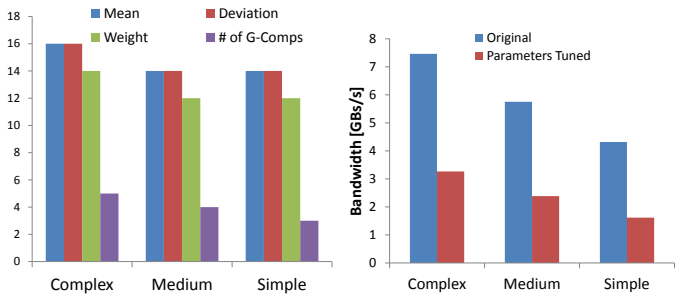
They track the multi-modal changes in background including variations in illumination and waving leaves caused by wind. In order to reduce memory bandwidth, we tune the width of individual Gaussian parameters. Fig. 7a also shows that storing *Mean* with 16-bits is sufficient. No quality improvement is achieved by storing the *Mean* with more than 16-bits. With 5 Gaussian components and 16-bit *Mean* in place, Fig. 7b presents the effect of *Weight* and *Deviation* width on output quality. It is characteristic that a plateau is reached early. 16bit *Deviation* as well as 14bit *Weight* already achieve the best quality.

Fig. 8a and Fig. 8b explore the same dependencies for the medium scene (subway station). Fig. 8a shows that a considerable quality improvement is achieved by increasing number of Gaussian components from 3 to 4. However no further quality improvement occurs when increasing to 5 Gaussian component. Similarly, the quality improvement saturates at 14bit *Deviation* and 12bit *Weight* as shown in Fig. 8b).

C. Results Generalization

Fig. 9 generalizes our findings for scenes of different complexity in terms of Gaussian parameter length (Fig. 9a) and resulting memory bandwidth (Fig. 9b). Highlighted in Fig. 9a, three, four and five Gaussian components are employed for *simple*, *medium* and *complex* scenes, respectively. *Medium* and *simple* are sufficiently covered with 14/14/12 bits (mean / deviation /weight), only the *complex* scene requires additional bits with 16/16/14 bits.

By reducing parameter bit-widths we significantly reduce memory bandwidth without visible quality loss and retaining ≥98% MS-SSIM. Fig. 9b shows the memory bandwidth reduction for different scene complexity in comparison to uncompressed 32 bit storage. The memory bandwidth is significantly reduced by up to 63% for the *simple* scene; from 4.3 MBytes/s to 1.6 MByte/s (63%). Bandwidths of medium and complex scenes are also reduced by 59% and 56% respectively. These huge reduction in communication demand come at a slight cost of increased computation. Compression/decompression increases overall computation by 6%. Since compression is not on a critical path, the latency of design does not necessarily increase.



(a) Gaussian parameters width (b) Required memory bandwidth

Figure 9: Summary of configuration and memory bandwidth

V. MoG HW Architecture

To derive a suitable HW architecture, we first explore parallelism, decomposition and hierarchy of our MoG specification model captured by SpecC SLDL.

The input MoG algorithm operated on a frame granularity. While this is still suitable for early estimation (and software implementation), it does not allow for pipelined implementation due to storage demands. Hence, we translated a frame-based MoG into to a pixel-based calculation allowing us to explore pipelined implementations. Fig. 10 outlines the top-level model of our MoG specification model. Gray pixels stream through "Gray Pixels" FIFO channel; RGB-to-Gray is considered as a pre-processing stage. Output foreground masks are also sent out through "FG Pixels" channel for further analysis including object detection and tracking.

We split our MoG model into three main behaviors: *Gaussian Components*, *BG Model Update* and *FG Detection* executing in a pipeline fashion on individual streaming pixels. Inside *Gaussian Components*, individual Gaussians execute in parallel independently from each other. Gaussian calculation and parameters update are independent among Gaussian components. Furthermore, specification model contains *Parameter Fetch Compression* and *Parameter Fetch Decompression* for reducing the stored bit-width of Gaussian parameters and thus reducing memory bandwidth pressure as we analyzed in Section IV.

Using our exploration results, we propose an architecture shown in Fig. 11 derived from our specification model (Fig. 10). Our MoG uses a 5 stage pipeline: (1) parameter fetch, (2)

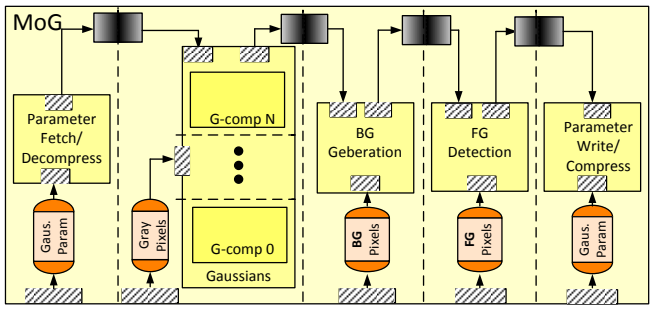


Figure 10: Pipelined MoG specification

Gaussian calculation, (3) BG modeling, (4) FG detection, parameter write-back. Operations in all 5 stages occur on individual pixels.

In stage one, parameter fetch reads Gaussian parameters through the system crossbar from memory as well as the input pixels from the camera stream. It decompresses the parameters into a common 32-bit format. At stage two, Gaussian calculation is performed in parallel for each component, it includes identifying match components and updating Gaussian parameters according to match result. Component calculation can further be pipelined internally. Stage three updates the background model of the current pixel, and stage four identifies a pixel as FG if no match component was found. Finally, stage five compresses the parameters and writes them back to memory. FG pixels are streamed to the host processor for a downstream vision applications (e.g. analysis).

Our HW-based MoG realization offers sufficient parallelism (across Gaussian components) and pipelining opportunities (within components) allowing dealing with the computation demand. Our MoG architecture offers configurability in number of components and parameter length to adjust to scene complexity or desired output quality. This enables to track a range of BG changes (e.g. light changes vs. rustling leaves) and adjusting the quality at cost of additional memory bandwidth (and computation).

VI. Conclusions

Tremendous opportunities open up when when jointly considering algorithm and architecture. In this paper, we have highlighted on an example of a Mixture-of-Gaussian (MoG) algorithm a methodological approach of joint algorithm tuning and system-level exploration. Our profiling results on MoG specification model captured in SpecC SLDL demonstrate an infeasibility of software implementation on embedded platforms. This drives the need for a HW-based implementation. In addition to the high computation demand, MoG also poses a very large volume of memory access per frame. Algorithm tuning (in this case parameter compression) is one approach to reduce memory bandwidth demands. Therefore, we propose a hardware-centric solution compresses parameters (with a configurable ratio) reducing the memory bandwidth at minor cost of some quality. We have reduced the memory bandwidth imposed by loading and storing Gaussian parameters by 63% at limited loss in quality (MS-SSIM $\geq$98%).

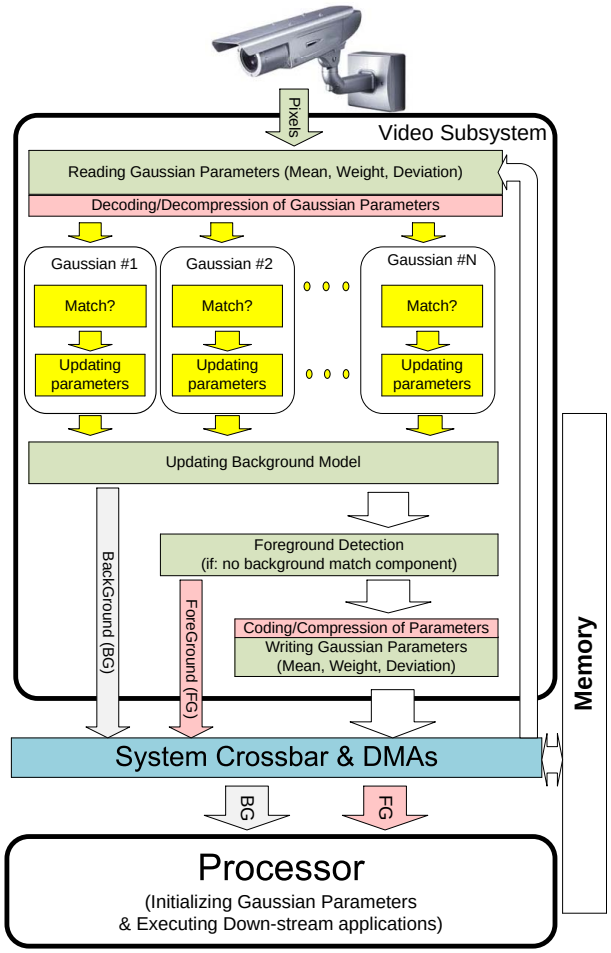


Figure 11: Architecture overview

References

[1] J. Bier, "Implementing Vision Capabilities in Embedded Systems," *Berkeley Design Technology Inc.*, Nov. 2012.

[2] A. R. Brodtkorb, C. Dyken, T. R. Hagen, J. M. Hjelmervik, and O. O. Storaasli, "State-of-the-art in heterogeneous computing," *Sci. Program.*, vol. 18, no. 1, Jan. 2010.

[3] Electronics Publications, "Video Surveillance Market: Global Forecast and Analysis 2011-2016," Available: http://www.electronics.ca.

[4] ABI Research, "Advanced Driver Assistance Systems Markets to Approach 10 Billion in 2011," Available: http://www.abiresearch.com.

[5] C. Stauffer and W. E. L. Grimson, "Adaptive background mixture models for real-time tracking," in *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*, vol. 2, 1999, pp. –252 Vol. 2.

[6] S.-C. Cheung and C. Kamath, "Robust techniques for background subtraction in urban traffic video," *SPIE Electronic Imaging*, 2007.

[7] M. Piccardi, "Background subtraction techniques: a review," in *Systems, Man and Cybernetics, 2004 IEEE International Conference on*, vol. 4, oct. 2004, pp. 3099 – 3104 vol.4.

[8] A. Gerstlauer, R. Dömer, J. Peng, and D. D. Gajski, *System Design: A Practical Guide with SpecC*, 2001.

[9] S. ching S. Cheung and C. Kamath, "Robust techniques for background subtraction in urban traffic video," *SPIE Electronic Imaging*, 2007.

[10] A. M. Elgammal, D. Harwood, and L. S. Davis, "Non-parametric model for background subtraction," in *Proceedings of the 6th European Conference on Computer Vision-Part II*, 2000, pp. 751–767.

[11] D. Comaniciu and P. Meer, "Mean shift: a robust approach toward feature space analysis," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 5, pp. 603–619, 2002.

[12] N. Oliver, B. Rosario, and A. Pentland, "A bayesian computer vision system for modeling human interactions," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 831–843, 2000.

[13] 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.

[14] M. Piccardi, "Background subtraction techniques: a review," in *IEEE International Conference on Systems, Man and Cybernetics*, vol. 4, Oct. 2004, pp. 3099–3104.

[15] R. Dömer, A. Gerstlauer, J. Peng, D. Shin, L. Cai, H. Yu, S. Abdi, and D. D. Gajski, "System-on-chip environment: a specc-based framework for heterogeneous mpsoc design," *EURASIP Journal of Embedded Systems*, vol. 2008, pp. 5:1–5:13, Jan. 2008.

[16] A. D. Inc.(ADI), "ADSP-BF60x Blackfin Processor Hardware Reference Manual," *Reference Guide, Part Number 82-100113-01*, Jun. 2012.

[17] *https://ece.uwaterloo.ca/ z70wang/research/ssim/index.html*.