Exploring Computation-Communication Tradeoffs in Camera Systems

Amrita Mazumdar*, Thierry Moreau*, Sung Kim†, Meghan Cowan*,
Armin Alaghi*, Luis Ceze*, Mark Oskin*, and Visvesh Sathe†
*Paul G. Allen School of Computer Science & Engineering, University of Washington
†Department of Electrical Engineering, University of Washington
{amrita, moreau}@cs.washington.edu, sungk9@uw.edu,
{cowanmeg, armin, luisceze, oskin}@cs.washington.edu, sathe@uw.edu

***Abstract*—Cameras are the defacto sensor. The growing demand for real-time and low-power computer vision, coupled with trends towards high-efficiency heterogeneous systems, has given rise to a wide range of image processing acceleration techniques at the camera node and in the cloud. In this paper, we characterize two novel camera systems that use acceleration techniques to push the extremes of energy and performance scaling, and explore the computation-communication tradeoffs in their design. The first case study targets a camera system designed to detect and authenticate individual faces, running solely on energy harvested from RFID readers. We design a multi-accelerator SoC design operating in the sub-mW range, and evaluate it with real-world workloads to show performance and energy efficiency improvements over a general purpose microprocessor. The second camera system supports a 16-camera rig processing over 32 Gb/s of data to produce real-time 3D-360° virtual reality video. We design a multi-FPGA processing pipeline that outperforms CPU and GPU configurations by up to 10× in computation time, producing panoramic stereo video directly from the camera rig at 30 frames per second. We find that an early data reduction step, either before complex processing or offloading, is the most critical optimization for in-camera systems.**

I. Introduction

Cameras are the backbone of data processing for applications ranging from social media and entertainment, to surveillance, biomedical devices, and autonomous vehicles. As these systems continue to specialize and diversify, the traditional interface between camera sensors and general-purpose processors limits optimization for extreme visual computing applications. Typically, architects employ one of two solutions to enable compute-heavy vision: on-device hardware acceleration, or cloud offload. Hardware accelerators achieve improved performance and efficiency at the cost of fixed functionality, while offloading data to the cloud relaxes computational constraints at the cost of data communication. The design tradeoff reduces to balancing computation and communication constraints for a given visual computing workload. In this paper, we investigate two end-to-end camera systems that push the boundaries of energy efficiency and performance, under these lenses of computation and communication costs. For each system, we focus on holistically evaluating the full system and computation-communication tradeoffs across parts of the processing system via "in-camera processing pipelines."

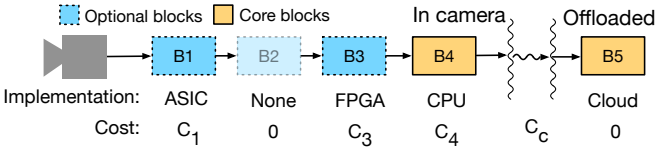


Fig. 1. Hypothetical in-camera pipeline with opportunities for acceleration. This pipeline uses *core* blocks and some *optional* blocks, and offloads computation to the cloud.

The first camera system is an ultra-low-power camera system that recognizes specific users' faces while running on harvested radio frequency (RF) energy. The ability to run untethered from a power source makes deployment simple, but pushes the design constraints to the extreme end of ultra-low-power design.

The second camera system assembles immersive stereoscopic virtual reality (VR) video processing in real time, requiring significantly more compute and communication performance. The system consists of 16 4K-resolution cameras, processing hardware, and a network link. To deliver real-time VR video, the system processes up to 32 Gb/s, making it impractical to transmit the sensor data to a data center for real-time stereo processing and stitching.

These systems push the bounds of camera system engineering, at opposing ends of the design space of camera systems—extreme low power, and extreme performance. Both systems decompose to pipelines of application-specific computation blocks, like the generic in-camera processing pipeline of Figure 1. By characterizing two extreme design points in a common framework, we highlight how these data movement considerations are common across the spectrum of camera applications. Many other camera systems are likely to exist between the power-constrained and high-bandwidth case studies we investigate.

This paper makes the following contributions:

- A low-power face authentication accelerator for energy harvesting cameras, with an ASIC evaluation on real-world workloads.
- A real-time stereoscopic video assembly accelerator for virtual reality, with a CPU, GPU and FPGA comparison.
- A joint evaluation of computation and communication costs, demonstrating how adding more computation can *reduce* the overall cost of accelerator-based image processing architectures.

II. Background and related work

In-camera processing is not new, and prior work has introduced many in-camera processors [17]. Our analysis applies in-camera processing pipelines to two highly-constrained applications: low-power face authentication and real-time VR video streaming. In this section, we discuss our general approach to analyzing image processing pipelines and review notable related work in computation offloading, image processing hardware, and similar accelerator designs.

In-camera processing pipelines. To characterize in-camera systems in a holistic way, we decompose camera applications into processing pipelines and evaluate the system at the level of functional block, as shown in Figure 1. Considering camera systems at the block granularity helps us gain insight into deciding what processing steps should be included at the camera node, and what implementations (e.g., ASIC, FPGA, GPU) meet an application's requirements. In the hypothetical pipeline of Figure 1, blocks B_1, B_3, and B_4 may be processed in-camera while the output of B_4 is offloaded to a central processor such as a multicore or cloud processor. The block B_2 is shown excluded from the pipeline because it does not improve the overall cost. We define the total cost of the pipeline as the sum of computation costs for in-camera blocks (C_1, C_3, and C_4) and the communication cost (C_c) of offloading the output of B_4. We assume the cost of computing in the cloud as "free" (relative to computation in the camera) but the cost to get data to the cloud is not (e.g., the camera expends energy to send data). Hence, one can view the main objective of computing in-camera is to minimize both the data communicated and the computational cost.

In-camera processing pipelines can include *core blocks* essential to the application, and *optional blocks*, which may not directly affect results but can improve efficiency by filtering or pre-processing data. One optional block is the motion detection block we use in our face authentication pipeline. While the core block of the pipeline, face authentication, operates on every input frame, an optional motion detection block can reduce the bandwidth and ensuing power consumption of core blocks.

Computation offload. Offloading image processing computation from mobile devices to the cloud is well-explored in mobile systems [33]. The opposing case for "onloading" computation, or keeping computation at the sensor, has grown more popular due to increased image processing demand and privacy concerns [16, 23]. Our approach explores the tradeoff space between offload and onload for two constrained camera systems.

Vision-centric architectures. The rise of computer vision and computational photography has inspired a number of computer architectures for efficient image processing. Flexible vision architectures [5, 7, 10] provide higher performance for image processing and vision applications while maintaining programmability. Mobile SoCs like Qualcomm's Snapdragon provide image processing functionality for mobile cameras [30]. Vasilyev et al. [38] argue towards programmable image processing solutions, but find that custom ASICs are still more energy efficient. Consequently, we choose to explore fixed-function hardware to meet the constraints of our ultra-low-power or high-performance application targets.

We consider different classes of image processing accelerators for the computational blocks in our case studies; we now detail related work in each class.

In-sensor processing. Image sensor data is typically captured as an analog signal and converted to a digital signal for processing. Recent work investigated how to improve application efficiency by moving some preliminary processing into the analog domain at the sensor node. Centeye, for instance, executes analog computation on image sensor signals [1]. Other work computed early layers of convolutional NNs at the pixel level [8, 22]. Processing can also be performed in the mixed-signal domain [2].

Face detection accelerators. We investigate the use of a face detection accelerator as an optional block to filter data in a face authentication pipeline. Hardware acceleration for the Viola-Jones face detection algorithm has been well-explored for FPGAs and GPUs [9, 18, 19]. While Bong et al. also present a neural network design using Haar filters as a first step, our work performs a more holistic characterization to optimize the full camera pipeline [6].

Neural network accelerators. NNs have been studied extensively for accomplishing face detection and recognition [14, 32, 35]. Researchers are actively working to improve NN performance with custom hardware [12, 13]. ShiDianNao [11], specifically, is a CNN accelerator executed in-camera, where the accelerator is placed on the same chip as the image sensor processor, achieving 320mW power consumption.

Depth from stereo accelerators. Depth from stereo algorithms and their implementations have been well-explored [34]. Stereo vision has been accelerated to real-time with GPUs and FPGAs, but application targets are either very lower resolution or perform badly on defocusing workloads [37, 44].

In-camera compression. Compressing sensor data incurs computation–communication tradeoffs related to this paper's analyses. In our VR pipeline, for instance, the output of some blocks might have a better data locality than the previous step, facilitating high compression rates, but lossy compression at the early stages of the pipeline could result in quality degradations. While we do not explicitly consider compression in our study, compression can be treated as an optional block in in-camera processing pipelines.

III. Case study: low-power face authentication

In this section, we characterize a continuous vision pipeline for face authentication based on the WISPCam platform [26], a battery-free camera powered by harvested energy. Face authentication (FA) is a core workload in user-centric continuous mobile

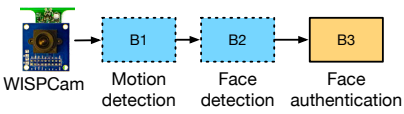


Fig. 2. Face authentication with battery-free cameras.

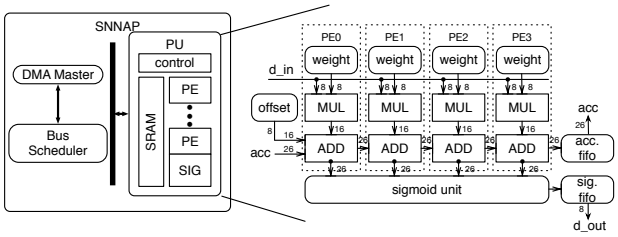


Fig. 3. NN microarchitecture and processing element details.

vision systems. In these systems, a camera captures image frames at a continuous frame rate, and an on-node processor performs face recognition on each frame to identify a single user. We define the core FA function as: given a test face and a reference, decide if the test face matches the reference face.

The WISPCam-based system captures an image at 1 frame per second (FPS) and transmits it over RF, powered by an internal capacitor with harvested RF energy. We examine how leveraging progressive filtering hardware can dramatically reduce the power consumption of such a system and enable continuous face authentication at low cost. We construct our FA pipeline around NN-based face authentication, as shown in Figure 2. The pipeline has one core block, the NN, and several optional blocks. We evaluate a low-power NN accelerator design, as well as the benefits of including motion detection and a pre-processing face detection accelerator to reduce input bandwidth to the NN. Because energy efficiency is a primary concern, we design the accelerators to be integrated on-chip with the camera sensor, and processed streaming through the CSI2 camera serial interface.

We first discuss each accelerator design individually, presenting their microarchitectures and the tradeoffs we investigated in each design's algorithm and hardware implementation. We then evaluate them together on a real-world face authentication workload using real video we collected.

A. Neural network face authentication

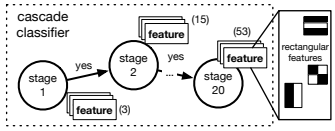
For our face authentication task, we investigate a systolic NN design, based on SNNAP, and explore tradeoffs in neural network (NN) topology, accelerator geometry, and datapath width reduction [25].

NN algorithmic tradeoffs. We first examine how modifying NN topology affects both classification accuracy and energy dissipation. We explore the search space by training NNs with Fast Artificial Neural Network Library [27] and measuring the achieved accuracy and energy cost.

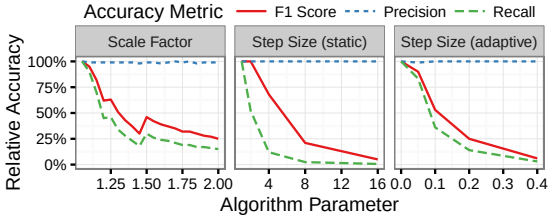
Increasing the number of layers and neurons directly impacts the memory and computational requirements of the NN. Varying the input size to the NN has a direct impact on performance and accuracy. Using a 5×5 low-resolution input window for face detection will lead to a cheap 25-neuron input layer, but results in poor accuracy. The largest input size our NN supports, 20×20 pixels, preserves more details, improving the accuracy of the NN classifier significantly. This comes at a cost: halving classification error incurs an order-of-magnitude increase in energy. From this exploration, we select the topologies that give us an optimal accuracy/energy

```
for x in range(0,image_width):
  for y in range(0,image_height):
    faces += classify(x,y,window)
    window *= scale_factor
    if window > image_size:
      return
```

(a) Algorithm pseudocode.



(b) Cascading classifier.



(c) Impact of VJ parameters on relative accuracy.

Fig. 4. The face detection algorithm slides a window across an image and repeatedly executes a classifier with stages of rectangular features.

compromise, a $400-8-1$ NN topology with 400 inputs neurons, 8 hidden neurons and 1 output neuron.

To evaluate accuracy tradeoffs, we trained a 400-8-1 NN on 90% of LFW [20], a popular face recognition benchmark, and tested its accuracy at recognizing a single person's face from the remaining 10%. Our evaluation indicates that with a $400-8-1$ topology, we can achieve a 5.9% classification error overall. As we discuss on our real-world evaluation, however, our multi-stage approach and real-data workload lowers the true miss rate of 0%, as the security workload presents many less-challenging lighting and orientation scenarios.

NN microarchitecture. Our NN microarchitecture uses a single processing unit with multiple processing elements. Because our face authentication pipeline has wide layers, we found that this design presented enough data parallelism to keep functional unit utilization high for a single processing unit. Figure 3 shows the datapath of a processing unit composed of four 8-bit processing elements (PEs). A bus connects the chain of processing elements to a sigmoid unit—a hardware LUT-based approximation of a neuron's activation function. Each PE has its own weight memory that stores the synaptic weights of the NN locally. The processing elements perform multiply-add operations in a systolic fashion to evaluate the matrix multiplication that composes each NN hidden and output layer. A vertically micro-coded sequencer sends commands to each processing element as inputs arrive and outputs are produced to control data movement.

The NN hardware accelerator has a configurable number of PEs, which we use to optimize the geometry of our accelerator. We fix the frequency and voltage to 30MHz and 0.9V, and explore the design tradeoffs between energy and throughput using post-synthesis physical simulations. We find an energy-optimal point at 8 PEs: any lower number of PEs introduces scheduling inefficiencies, increasing energy consumption; too many PEs results in underutilized resources and reduced parallelism for the narrow network.

NN numerical accuracy tradeoffs. Power dissipation in the memory and the PEs can be reduced by bit-width reduction. We used fixed-point functional units and LUT-based approxi-

mations of mathematical functions to minimize power and area. We study the impact of two precision knobs on application accuracy: (1) sigmoid approximation and (2) data bit-width. We evaluate error as absolute classification accuracy loss relative to a NN implemented with floating-point arithmetic and precise mathematical functions. We then evaluate fixed-point precision, limiting ourselves to powers of two for memory alignment.

After examining the effect of approximating the sigmoid function with a simple 256-entry look-up table (LUT), we conclude that hardware approximation of the sigmoid function has a negligible effect on accuracy. For datapath width, both 16-bit and 8-bit implementations of the NN accelerator result in a small 0.4% accuracy loss relative to a precise floating-point implementation. The 4-bit datapath however displays a significant accuracy loss on average (over 1%). The reduction in datapath width from 16-bit to 8-bit leads to a 41% power reduction for an 8-PE configuration, so we select 8-bit datapaths as the optimal energy-accuracy point for our NN implementation.

B. In-camera face detection

The Viola-Jones (VJ) face detection algorithm is a popular computer vision algorithm for fast, accurate face detection [40]. It is widely used in face authentication and other situations where frontal faces are expected and speed is preferred. The algorithm detects faces by scanning a window across the image, evaluating simple rectangular features within the window at each window position. If enough of these features are found at a single window position, then that window is identified as a face. To account for faces of different sizes in an image, the scanning window is scaled and passed over the scene multiple times. The VJ algorithm is well-known because of its simplicity and efficiency, and continues to perform well against more complex algorithms including deformable parts models and convolutional NNs on face detection [24].

The VJ algorithm is popular specifically because of its high efficiency in non-face windows – the algorithm optimizes to spend more computation on windows where there is likely to be a face, rather than executing a uniform computation at every window. This optimization is encoded in the *cascade classifier* structure illustrated in Figure 4b, a nested decision tree where progressive levels have increasingly more features to evaluate, and the simple stages must be evaluated positively first before continuing on. The cascading computational style makes VJ a good fit for a pre-filtering accelerator.

VJ algorithm tradeoffs. We first tune the parameters of the VJ algorithm in search of the optimal point in the energy-accuracy tradeoff space. VJ classifiers are structured to minimize computation in non-face windows, and executing a full positive face detection incurs a larger computation cost. The parameters of *window scale factor*, the size of the window scanned looking for a face, and *window step size*, the space between windows evaluated over the image, affect the number of kernel invocations, as shown in Figure 4a. Varying these parameters of window step size and window scale factor directly impacts the energy and accuracy of this block. Conventional

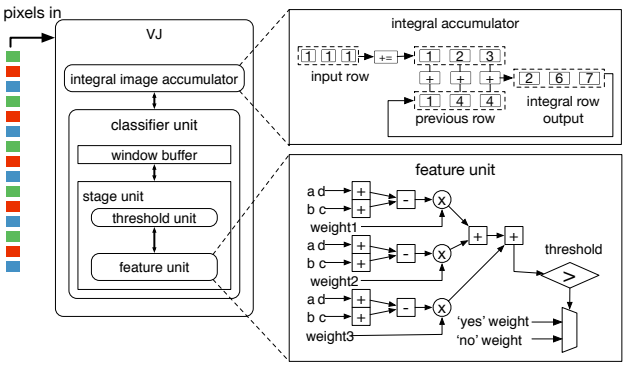


Fig. 5. Proposed Viola-Jones accelerator architecture.

implementations use a scale factor of 1.1 and a step size of 1, which is inefficient at large window sizes when the features are more insensitive to small changes in translation and scale. We varied the scale factor between 1.05 and 2 and the window step size between 1 and 16 and characterized how they impacted accuracy and number of invocations. We also evaluated an *adaptive step size*, where the step size is a percentage of the window size, rather than a fixed value. Figure 4c shows our results, where accuracy is normalized to the most fine-grained parameter choice. We report accuracy via *precision*, the ratio of correctly classified faces over all samples classified as a face, *recall*, the ratio of classified faces over all true faces, and *F1 score*, the harmonic mean of precision and recall. We find that varying these parameters affects recall but not precision, and thus choose a window scale factor of 1.25 and an adaptive step factor of 2.5%. This parameter selection results in 86% less invocations of the VJ classifier and no loss in accuracy for our real-world workloads.

VJ streaming microarchitecture. A high-level block diagram of the microarchitecture is shown in Figure 5. The accelerator design consists of two functional modules: (i) the integral image accumulator, which transforms the image for easy feature extraction, and (ii) the cascade classifier implementation, which evaluates features and computes the classifier result for an image window.

Many VJ accelerators exist in the literature for different application domains and power envelopes. The key observation that facilitates sub-mW power consumption for our accelerator is to process the data in a streaming fashion, leveraging the nature of the pixel stream. Classical VJ accelerator designs store the input image in memory and compute the integral image either in-place or writing it as a new image. Instead, we design a functional unit to compute the integral image in a stream-like pattern. Our integral image unit buffers each row, adding it to a "last row" buffer and also consecutively adding each pixel to its neighbor, as illustrated in Figure 5. By computing the integral image for a whole image while buffering just two rows, we use significantly less storage than if we had buffered the whole image to compute the integral image result: for our WISPCam workload, we use less than 1kB to hold the necessary rows for integral image output, whereas buffering the whole image would require a 57kB buffer. We

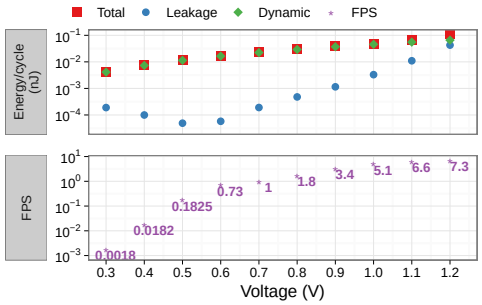


Fig. 6. Impact of voltage scaling on energy consumption and frame rate of the face authentication SoC.

also included a unit to transform face windows from integral image form to standard images before they are transferred to later stages of a camera pipeline.

C. Physical implementation and optimization

We implemented and synthesized our designs in Verilog with the Synopsys Design Compiler and TSMC 65nm Gplus Standard VT library, and performed place-and-route with the Synopsys IC Compiler. We generated SRAM macros with an ARM Advantage memory compiler. To obtain accurate energy estimates, we measured the initiation interval of the hardware pipelines in functional simulation and used power dissipation numbers from PrimeTime-PX to derive energy. To find the optimal system voltage, we derived the static and dynamic components of the total system energy and selected the minimum-energy voltage that still satisfies system performance requirements. Because application's performance requirement of 1 FPS allowed the design to run slower than the nominal frequency at 0.9V, we scaled the system voltage to sub-threshold and derived the optimal voltage for the entire SoC. We used the sub-threshold leakage-current relationship from [42] and approximate system frequency based on prior low-voltage 65nm SRAM implementations [29]. Figure 6 shows that there is a leakage energy minima at 0.5V, but the dynamic and total energies continue to decrease into the sub-threshold region. Hence, we considered the optimal system voltage to be the minimum voltage satisfying performance constraints. For our application, we selected the 0.7V/28 MHz operating point, which satisfies the WISPCam's application performance constraint of 1 FPS.

D. Experimental results

Table I describes the details of our evaluation. We compared the performance and energy of our accelerator configuration against a software baseline running on an MSP430, a low-power microprocessor on the WISPCam. We wrote an NN micro-benchmark trained for face authentication, and emulated execution on a synthesized OpenMSP430 design. We then compared the performance of the NN and face detection (FD) accelerators.

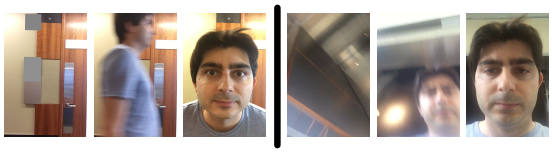


Fig. 7. Frames from our face authentication workloads.

In our evaluation, system requirements constrain the baseline choice: the MSP430 is the only processor operating within the power constraints of current energy-harvesting systems. Higher-performance embedded processors, while able to execute our application with better performance than an MSP430, were limited by the power budget required. In prior NN-accelerator work, the most appropriate comparison is ShiDianNao [11], which also operates at ∼300mW, still significantly higher-power than our design.

Improvements over a single frame. We examined three scenarios: (1) face authentication in software with the MSP430, (2) face authentication solely with the NN accelerator and (3) face authentication NN and FD accelerators. As expected, hardware acceleration leads to substantial performance improvement over a software implementation on an OpenMSP430 alone, and that coupling the FD and NN accelerators further improves both energy efficiency and speed. Specifically, hardware acceleration results in 265× speedup and 442.146× energy savings over software implementations of face recognition. Much of this speedup can be attributed to parallelization; the MSP430 can only run single-threaded applications, whereas the NN accelerator can take advantage of the regular, intrinsic parallelism of neural networks. Moreover, we carefully tune the storage requirements of our accelerators, reducing leakage energy and consequently overall energy.

Tradeoffs in computation and communication. To consider the computation and communication power of our accelerators and the CPU baseline, we evaluated all configurations of the face authentication pipeline on experimentally-collected videos of real-world workloads. The dataset contains video scenarios we crafted to be representative of mobile face authentication in the wild, primarily surveillance-style security videos of a lab and a videos from a wearable-style device with an always-on camera. To match the system design requirements of an energy-harvesting platform, we captured and processed the video at 1 FPS. Figure 7 shows select frames.

Using these benchmarks and the power/performance characteristics of our accelerators, we derived the input bandwidth and resulting power consumption for each computational block in every configuration of the system. To derive the cost to offload image data, we used previously reported communication power numbers [26]. Figure 8 shows our results for the full system of image sensor capture, motion detection, FD, and NN face authentication, comparing total power consumption on combinations of the ASIC designs described in this section. Because the CPU could not compute the face recognition kernel on even one 400-pixel window at 1 FPS, we did not consider the results in this section. The configurations including CPU face authentication consume 2-5 orders of magnitude more power overall than the full-ASIC design we implemented.

Figure 9 shows the detailed power breakdown between computation and communication for the full system in silicon. The computation power is the sum of power at that block and

TABLE I
FACE AUTHENTICATION SYSTEM PARAMETERS.

Experimental Parameters		VJ Accelerator		NN Accelerator		OpenMSP430 [28]	
Technology	65nm TSMC GP HVT	Memory	1kB frame buffer	Memory	512kB IM, 4kB DM	Memory	2kB IM, 2kB DM
Tool Chain	Synposys	Datapath	16-bit custom logic	Datapath	8×8-bit PEs	ALU	16-bit with multiply
C Compiler	TI msp430-gcc [21]	Cascade	10×33	Sigmoid	256B LUT	Power	181µW
SRAM	ARM compiler + derivation	Input	176×144 pixel stream	Input	400-pixel image window	Area	$0.12mm^2$
Vdd	Low-voltage, 0.7V	Power	337µW	Power	393µW		
Freq	27.9MHz	Area	$0.06mm^2$	Area	$0.38mm^2$		

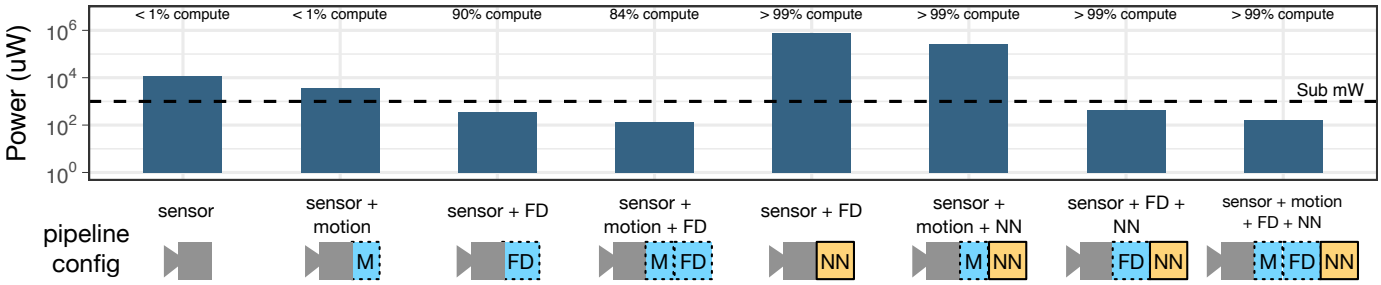


Fig. 8. The total power for different ASIC accelerator configurations is typically dominated by either computation or communication power. The lowest-power solution uses the motion and face detection filters and offloads the NN.

the processing blocks preceding it, and the communication power is the cost to transfer the output of that block. As expected, the computation power increases as more blocks process the data while the communication power decreases, but not at the same rate. Counterintuitively, the total power increases by 28% after performing the NN, indicating that *it is more power-efficient to offload after motion and face detection, rather than performing the full pipeline.* Even though the NN only needs to communicate a 1-bit response, the cost of computation increases dramatically in comparison to the decrease in communication power. This result indicates that it is more cost-effective to offload the neural network than process it in-camera, in this power-constrained application domain.

So, why not always offload neural networks? We examined the extent to which different constraints in our design contributed to our results. We found that if the communication power cost per pixel grew by 2.68×, it would be more power-efficient to perform the NN in-camera. Our evaluation sourced communication power data from work using a directed RFID reader, and RF-based energy harvesting systems may not always deliver consistent power.

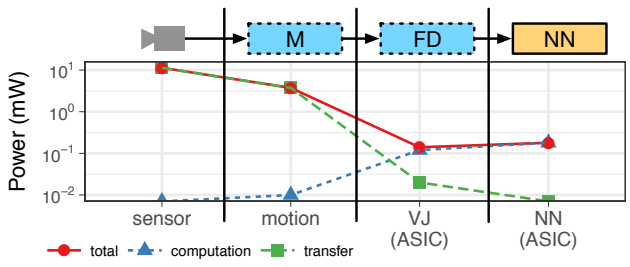


Fig. 9. Computation and communication power after each ASIC block in the "full pipeline" configuration .

Analysis of the video workloads highlights the benefits and limitations of our filtering approach at the application level. Importantly, the complete pipeline, consisting of motion detection, VJ, and NN, did not eliminate any true faces in any of our workloads. Using these filtering steps reduces data bandwidth, but we find that the extent of the data reduction could be improved. Motion detection often is triggered in innocuous situations, such as when people pass by in a frame or a mobile device is carried while walking. Our face detector misclassifies many false positives as faces, and also detects faces in static posters and photos. We examined the results of one of our security authentication workloads to illustrate: out of 62 frames of video, 12 frames were accepted through a motion detection block. In those 12 frames, the VJ detector passed forty 400-pixel face windows to the NN classifier. Visual inspection of those 40 faces found that 10% were false positives—with a perfect face detector that only detected true faces, the power to offload or compute the NN on-device would be reduced.

To investigate the impact of image size on the tradeoff between in-camera and offloading, we scaled our evaluation from the low-resolution images produced by the WISPCam to high-resolution mobile camera images. We found that keeping the neural network in-camera became power-efficient at 8 megapixels or greater. Finally, while offloading data may be more power-efficient, privacy and bandwidth concerns may lead designers to opt for a more localized solution when processing image data for face authentication tasks. These results indicate that as mobile camera devices become more sophisticated, architectures will have to become increasingly specialized to deliver low-power in-camera results.

IV. CASE STUDY: REAL-TIME VIRTUAL REALITY VIDEO

In this section, we investigate the use of in-camera processing for a high-performance, real-time panoramic stereo video rendering application. As shown in Figure 10, the pipeline

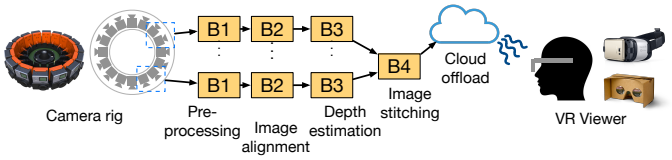


Fig. 10. 3D-360° virtual reality video generation, capture and viewing devices.

we consider takes as input the high-resolution camera feeds from a rig of cameras, like Google Jump [3], and processes the images into a 360° stereo pair viewed on a VR viewer, such as Google Cardboard [15]. The goal is to produce high-quality video streams at a frame rate of 30 frames/sec or more.

Many VR video pipelines pipelines require users to upload camera streams to a cloud service or high-performance computing system—this workflow prevents real-time applications such as live VR video streaming. While real-time hardware systems for processing VR video are becoming commercially available [36, 39], these solutions provide either live panorama processing or stereoscopic 3D, not both. In our design, we evaluate the performance constraints of this multi-step pipeline and investigate how much in-camera processing is required to achieve real-time VR video generation. We evaluate how processing at the camera node reduces the bandwidth required for offloading, and how hardware acceleration facilitates a real-time VR system.

Camera rigs for recording stereoscopic panorama videos capture a multi-camera scene and compute a depth map for each pair of cameras in the rig. These depth maps are composited together from multiple pairwise-camera pipelines into a single 3D-360° video. For our application, we seek to meet a real-time frame rate of 30 frames/sec, so we optimize our design for the cost of throughput. We define the communication cost as the bandwidth in and out of each block. Since all the pipeline blocks and offloading can be pipelined, the slowest step will dominate overall throughput. Among the blocks shown in Figure 10, the depth estimation step has the lowest bandwidth and throughput. In this section, we describe the depth estimation algorithm used for this block, how we map the algorithm

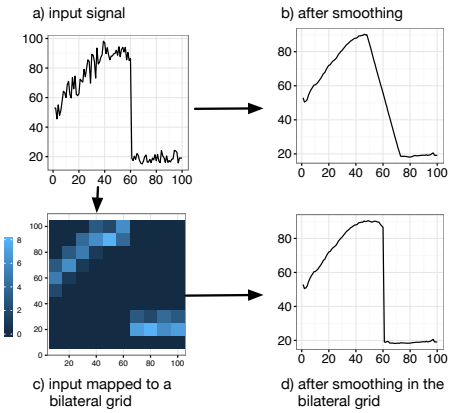


Fig. 11. The bilateral filter is an edge-aware filter.

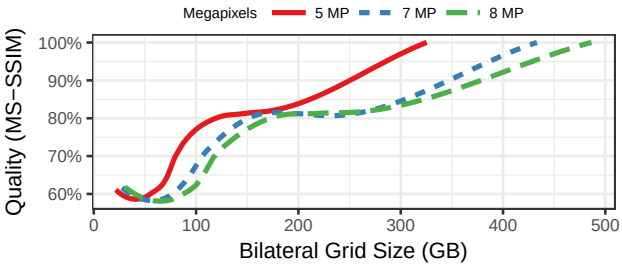


Fig. 12. Using a smaller bilateral grid is cheaper to compute but degrades the quality of the output depth map, even at high image resolutions.

to a high-throughput accelerator, and evaluate the system's computation-communication tradeoffs towards real-time results.

A. *Depth maps from bilateral-space stereo*

We base our design for fast and accurate stereo processing on the state-of-the-art bilateral-space stereo algorithm (BSSA) [4]. Typically, global stereo algorithms generate a depth map from a pair of images by computing a rough disparity, or difference in space, between pixels, and then refining that disparity until a cost function has been minimized [34]. Instead of computing disparities per-pixel, BSSA resamples the problem into a different representation, *bilateral-space*, before computing the disparity. In the bilateral domain, simple local filters are equivalent to costly, global edge-aware filters in pixel-space—consequently, disparity refinement is much faster in bilateral space. We perform BSSA in a *bilateral grid* data structure, where pixels are mapped to a grid vertex, or bin, in bilateral-space. Filtering in the bilateral grid results in faster, higher-quality output than comparable techniques [4].

We illustrate the operation of a bilateral filter in Figure 11. For simplicity, we demonstrate a 1D signal, instead of a 2D image signal. Our stereo algorithm seeks to smooth the noisy signal of Figure 11a, which has a sharp edge. Applying a 1D moving average on Figure 11a results in Figure 11b, which has less noise and a smoothed-out edge. A bilateral filter performs the same smoothing operation while preserving the edge of Figure 11a. The signal is mapped to bilateral space as in Figure 11c, where neighboring pixels with significantly different intensity values will have a large distance in 2D-space. Smoothing this signal in the bilateral domain with a 2D moving average allows the signal to maintain edges. Figure 11d shows the result after filtering in bilateral-space.

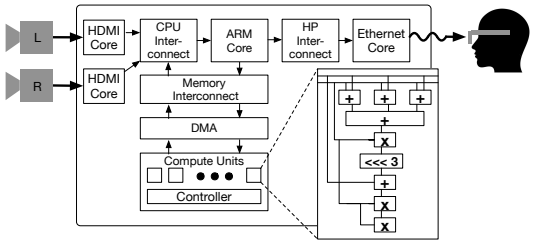


Fig. 13. VR accelerator architecture on a Xilinx Zynq SoC.

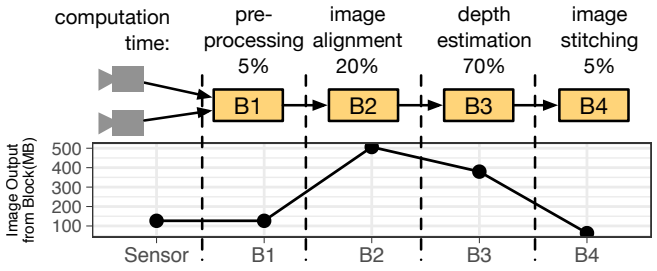


Fig. 14. Computation distribution and output data size for blocks in a VR video pipeline (2 of 16 cameras).

Instead of a simple filter like moving average, BSSA maps a noisy depth map to a bilateral grid, refines the depth map by solving an optimization problem, and remaps the bilateral-grid result to pixel-space. Varying the number of pixels that map to a grid vertex impacts the time to compute the stereo refinement for a frame, and also the quality of the depth map. Figure 12 demonstrates the tradeoff between stereo image quality and bilateral grid size to be processed for high-resolution input images. Here, we scaled bilateral grid sizes from 4 pixels-per-grid-vertex to 64 in each of three dimensions in a bilateral grid and evaluated the resulting impact on quality using MS-SSIM [41]. We find the resolution of the input images is less impactful than choosing an appropriate grid size to balance quality and computational complexity.

B. BSSA accelerator design on FPGAs

We design and implement our processing flow in Verilog on the Xilinx Zynq-7020 SoC [43]. Figure 13 depicts the high-level architecture of our system. We implement the initial full pipeline in software to run on the Zynq's CPU, and then design an AXI-Stream-compliant FPGA accelerator for depth refinement that can be invoked by the software. The CPU prepares the bilateral-grid data structure with pixels mapped to grid vertices, and transfers them via DMA to the FPGA fabric. The hardware accelerator processes the vertices with the bilateral-space filtering and streams them back to the CPU, where the bilateral-grid-filtered result is converted into the fully-processed depth map.

Figure 14 shows the processing break-down for our pipeline in time consumption and the image data size produced by each block. We find that the depth estimation block, B_3, consumes the greatest computation time as well as the largest amount of data, from B_2. We thus focus on applying FPGA acceleration to this block, and then evaluate the impact of accelerating this block on pipeline throughput.

Applying the computation of B_3 to a high-resolution video is equivalent to applying millions of blurs to the bilateral grid representation of the video frames. Across a single frame, most of these filters can run in parallel, so we designed streaming compute units to run bilateral filters on a stream of grid vertices. We find that BSSA requires at least 32-bit floating-point precision to produce high-quality depth maps, and use DSP units on the FPGA fabric to compute efficient floating-point operations. Each compute unit requires 18 DSP units in our design, so we can scale up to 12 parallel compute units on the ZC702. However, we project that if we scale up to a top-of-the-line Xilinx Virtex UltraScale+ FPGA, we can parallelize up to 682 compute units, which are more than enough for real-time operation. Table II summarizes the setup we use in our evaluation and resource requirements for real-time performance with a 16-camera system.

TABLE II
REQUIREMENTS FOR FPGA ACCELERATION PLATFORM.

	Resource	Evaluation	Target
System	FPGA Model	Zynq-7000	Virtex UltraScale+
	FPGA (#)	1	16
	Cameras	2	16
Per FPGA	Logic	45.91%	67.10%
	RAM	6.70%	17.60%
	DSP	94.09%	99.98%
	Clock (MHz)	125	125

C. Evaluation

Experimental setup. We compare our FPGA results on the Zynq platform to CPU and GPU baselines. The Zynq includes a Dual ARM Cortex-A9 and a Xilinx FPGA, all fabricated at TSMC 28nm technology. We implement the CPU baseline on the Zynq's Dual ARM Cortex-A9 as a proxy for a mobile-grade CPU, and evaluate the GPU on an NVIDIA Quadro K2200. Both baselines execute optimized BSSA code written and tuned with Halide [31].

Methodology. We consider the throughput of the data output as the "communication cost" for offloading, and the cost to compute the pipeline block as the "computation cost". We treat the communication cost as fixed for each block; it is simply the cost of offloading the data from each block, as shown in Figure 14. For all blocks except disparity refinement, we assume the computation cost to be the compute time evaluated using the ARM CPU baseline's performance numbers. We average the compute time for the disparity refinement block over five executions of the kernel over a frame. Because this processing flow can be pipelined across frames in a video stream, the "total cost" of the system can be considered to be dominated by the lowest-throughput block of the system.

Computation-communication tradeoffs. Figure 15 shows the runtime results of different pipeline configurations, uploaded on a networked connection to a viewing device supporting at least 30 FPS. We seek to uncover scenarios in which both computation and communication surpass our minimum frame rate of 30 FPS—if one or both costs falls below the threshold, the system cannot support real-time operation.

For the first three scenarios, the cost of doing little computation before offloading is cheap, even on the ARM core, but the communication cost for the raw captured data falls short of our 30 FPS threshold. Computing the disparity refinement in B_3 is more costly, and the CPU and GPU implementations are not fast enough to support real-time operation. Moreover,

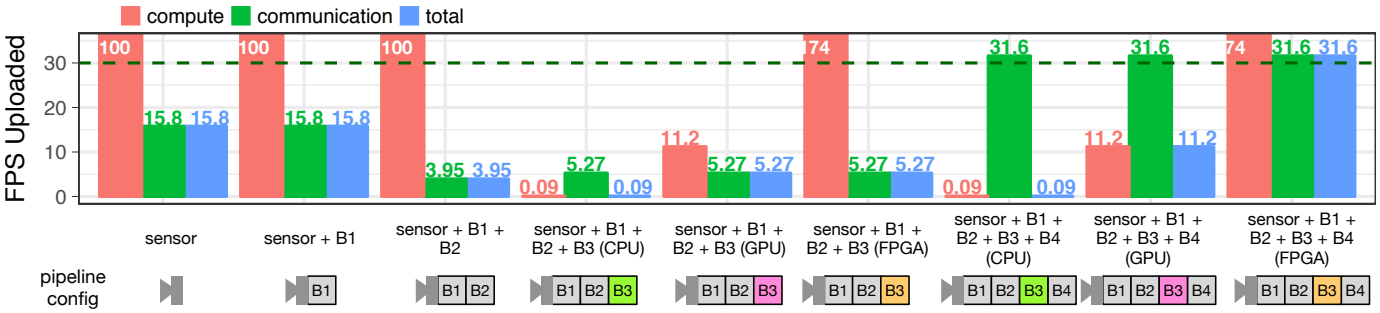


Fig. 15. Pipeline configurations with different bilateral smoothing implementations (CPU, GPU, FPGA), and resulting upload rates (frames per second). Only the full pipeline with FPGA acceleration can meet a 30 FPS upload requirement.

the cost of offloading the computed depth maps before image stitching is significantly lower.

The computation cost of image stitching in B_4 is marginal compared to BSSA, as well, and the resulting FPS is virtually the same. The data size to communicate after B_4, however, is much smaller, as illustrated in Figure 14, and is the only data size small enough to support real-time uploading. We find that the configuration with all the blocks processed in-camera and B_3 mapped to the FPGA is the only configuration where both computation and communication pass the threshold and support real-time processing.

Our analysis indicates that this camera system is primarily constrained by network bandwidth. For our evaluation, we assumed transfer speeds of 25 Gigabit Ethernet. As network connections grow faster, our results will trend towards off-loading computation right off the sensor. For instance, at a hypothetical ultra-high-throughput network link of 400-Gb Ethernet, the 16-camera output can be uploaded at 395 FPS, reducing the efficiency incentive for in-camera processing in this scenario.

V. Conclusions

Cameras have become the dominant sensor in mobile systems, and complex image processing pipelines are now standard. In this paper, we use the notion of "in-camera processing pipelines" to thoroughly characterize the design of two camera systems at the extreme ends of the energy and performance scaling limits of current hardware. Our face authentication camera system, for instance, runs entirely on harvested energy, pushing the limits of ultra-low power computation. Our virtual reality camera system requires significantly more in-camera processing and data communication resources than traditional imaging platforms. Our results highlight how design parameters for individual accelerators can influence the full-system execution behavior, as well as shape decisions about whether to process a computation block in the camera or offload the computation.

We characterize in detail how even the most power-efficient neural network design performs significantly better when adding computation earlier in the pipeline to effectively filter the image data. Our VR pipeline highlights how computational stages that expand the data size are inefficient in isolation, and can be better optimized in concert with their down-stream components.

Power and performance constraints require increasingly efficient computational platforms, and architects will continue to look to hardware acceleration to enable challenging applications. As we demonstrate in this paper, even tightly-optimized accelerators can fail to improve performance if they fail to consider full-system communication challenges. Given the growth of image data production and requirements of modern vision and graphics algorithms, future applications require a full system approach to maintain power and performance efficiency of camera designs.

VI. Acknowledgements

We thank members of the Sampa lab and the anonymous reviewers for their feedback on earlier versions of this work. This work was supported in part by the National Science Foundation under Grant CCF-1518703, by C-FAR, one of the six SRC STARnet Centers, sponsored by MARCO and DARPA, and a generous gift from Google.

References

[1] "Vision chips," http://www.centeye.com/technology/vision-chips/, accessed: 2017-06-06.

[2] A. Alaghi, C. Li, and J. Hayes, "Stochastic circuits for real-time image-processing applications," *DAC*, 2013.

[3] R. Anderson, D. Gallup, J. T. Barron, J. Kontkanen, N. Snavely, C. Hernández, S. Agarwal, and S. M. Seitz, "Jump: Virtual reality video," *SIGGRAPH Asia*, 2016.

[4] J. T. Barron, A. Adams, Y. Shih, and C. Hernandez, "Fast bilateral-space stereo for synthetic defocus," *CVPR*, 2015.

[5] B. Barry, C. Brick, F. Connor, D. Donohoe, D. Moloney, R. Richmond, M. O'Riordan, and V. Toma, "Always-on vision processing unit for mobile applications." *IEEE Micro*, vol. 35, no. 2, 2015.

[6] K. Bong, S. Choi, C. Kim, S. Kang, Y. Kim, and H.-J. Yoo, "A 0.62 mw ultra-low-power convolutional-neural-network face-recognition processor and a cis integrated with always-on haar-like face detector," *ISSCC*, 2017.

[7] N. Chandramoorthy, G. Tagliavini, K. Irick, A. Pullini, S. Advani, S. A. Habsi, M. Cotter, J. Sampson, V. Narayanan, and L. Benini, "Exploring architectural heterogeneity in intelligent vision systems," *HPCA*, 2015.

[8] H. G. Chen, S. Jayasuriya, J. Yang, J. Stephen, S. Sivaramakrishnan, A. Veeraraghavan, and A. Molnar, "Asp

vision: Optically computing the first layer of convolutional neural networks using angle sensitive pixels," *CVPR*, 2016.

[9] J. Cho, B. Benson, S. Mirzaei, and R. Kastner, "Parallelized architecture of multiple classifiers for face detection," *ASAP*, 2009.

[10] J. Clemons, C.-C. Cheng, I. Frosio, D. Johnson, and S. W. Keckler, "A patch memory system for image processing and computer vision," *IEEE/ACM Microarchitecture*, 2016.

[11] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "ShiDianNao: Shifting vision processing closer to the sensor," *ISCA*, 2015.

[12] C. Farabet, B. Martini, P. Akselrod, S. Talay, Y. LeCun, and E. Culurciello, "Hardware accelerated convolutional neural networks for synthetic vision systems," *ISCAS*, 2010.

[13] C. Farabet, B. Martini, B. Corda, P. Akselrod, E. Culurciello, and Y. LeCun, "NeuFlow: A runtime reconfigurable dataflow processor for vision," *CVPRW*, 2011.

[14] C. Garcia and M. Delakis, "Convolutional face finder: A neural architecture for fast and robust face detection," *PAMI*, 2004.

[15] Google, "Cardboard - Google VR," https://vr.google.com/cardboard/, accessed: 2017-06-06.

[16] S. Han and M. Philipose, "The case for onloading continuous high-datarate perception to the phone," *HotOS*, 2013.

[17] J. Hauswald, T. Manville, Q. Zheng, R. Dreslinski, C. Chakrabarti, and T. Mudge, "A hybrid approach to offloading mobile image classification," *ICASSP*, 2014.

[18] D. Hefenbrock, J. Oberg, N. T. N. Thanh, R. Kastner, and S. B. Baden, "Accelerating Viola-Jones face detection to fpga-level using gpus," *FCCM*, 2010.

[19] M. Hiromoto, K. Nakahara, H. Sugano, Y. Nakamura, and R. Miyamoto, "A specialized processor suitable for AdaBoost-based detection with Haar-like features," *CVPR*, 2007.

[20] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller, "Labeled faces in the wild: A database for studying face recognition in unconstrained environments," University of Massachusetts, Amherst, Tech. Rep. 07-49, 2007.

[21] T. Instruments, "Msp430-gcc - open source compiler for msp430 microcontrollers," http://www.ti.com/tool/msp430-gcc-opensource, 2014-2015.

[22] R. LiKamWa, Y. Hou, J. Gao, M. Polansky, and L. Zhong, "Redeye: Analog convnet image sensor architecture for continuous mobile vision," *ISCA*, 2016.

[23] R. LiKamWa, Z. Wang, A. Carroll, F. X. Lin, and L. Zhong, "Draining our glass: An energy and heat characterization of google glass," *APSys*, 2014.

[24] M. Mathias, R. Benenson, M. Pedersoli, and L. Van Gool, "Face detection without bells and whistles," *ECCV*, 2014.

[25] T. Moreau, M. Wyse, J. Nelson, A. Sampson, H. Esmaeilzadeh, L. Ceze, and M. Oskin, "SNNAP: Approximate computing on programmable SoCs via neural acceleration," *HPCA*, 2015.

[26] S. Naderiparizi, Y. Zhao, J. Youngquist, A. P. Sample, and J. R. Smith, "Self-localizing battery-free cameras," *UbiComp*, 2015.

[27] S. Nissen, "Implementation of a fast artificial neural network library (FANN)," Department of Computer Science University of Copenhagen (DIKU), Tech. Rep., 2003, http://fann.sf.net.

[28] OpenCores, "Openmsp430," http://opencores.org/project,openmsp430, accessed: 2017-06-06.

[29] M. Qazi, M. Sinangil, and A. Chandrakasan, "Challenges and directions for low-voltage SRAM," *IEEE Design & Test of Computers*, vol. 28, no. 1, 2011.

[30] Qualcomm, "Snapdragon," https://www.qualcomm.com/products/snapdragon/processors, accessed: 2017-06-06.

[31] J. Ragan-Kelley, C. Barnes, A. Adams, S. Paris, F. Durand, and S. Amarasinghe, "Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines," *PLDI*, 2013.

[32] H. A. Rowley, S. Baluja, and T. Kanade, "Neural network-based face detection," *PAMI*, 1998.

[33] M. Satyanarayanan, "Pervasive computing: vision and challenges," *IEEE Personal Communications*, vol. 8, no. 4, 2001.

[34] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *IJCV*, 2002.

[35] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "Deepface: Closing the gap to human-level performance in face verification," *CVPR*, 2014.

[36] Teradek, "Sphere - real-time 360 monitoring and live streaming," http://teradek.com/collections/sphere-family/, accessed: 2017-06-06.

[37] C. Ttofis and T. Theocharides, "High-quality real-time hardware stereo matching based on guided image filtering," *DATE*, 2014.

[38] A. Vasilyev, N. Bhagdikar, A. Pedram, S. Richardson, S. Kvatinsky, and M. Horowitz, "Evaluating programmable architectures for imaging and vision applications," *IEEE/ACM Microarchitecture*, 2016.

[39] Videostitch, "Vahana vr," http://www.video-stitch.com/live-vr/, accessed: 2017-06-06.

[40] P. Viola and M. J. Jones, "Robust real-time face detection," *IJCV*, 2004.

[41] Z. Wang, E. P. Simoncelli, and A. C. Bovik, "Multiscale structural similarity for image quality assessment," *IEEE Asilomar*, vol. 2, 2003.

[42] N. Weste and D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*, 4th ed. Addison-Wesley Publishing Company, 2010.

[43] Xilinx, "SoCs & MPSoCs," http://www.origin.xilinx.com/products/silicon-devices/soc.html, accessed: 2017-06-06.

[44] Q. Yang, "Hardware-efficient bilateral filtering for stereo matching," *PAMI*, 2014.