Hardware Acceleration of Feature Detection and Description Algorithms on Low-Power Embedded Platforms

Onur Ulusel, Christopher Picardo, Christopher B. Harris, Sherief Reda and R. Iris Bahar
School of Engineering
Brown University, Providence, RI 02912
{onur_ulusel, christopher_picardo, christopher_harris, sherief_reda, iris_bahar}@brown.edu

***Abstract*—Image features are broadly used in embedded computer vision applications, from object detection and tracking to motion estimation and 3D reconstruction. Efficient feature extraction and description are crucial due to the real-time requirements of such applications over a constant stream of input data. High-speed computation typically comes at the cost of high power dissipation, yet embedded systems are often highly power constrained, making discovery of power-aware solutions especially critical for these systems. In this paper, we present a power and performance evaluation of three low cost feature detection and description algorithms implemented on various embedded systems (embedded CPUs, GPUs and FPGAs). We show that FPGAs in particular offer attractive solutions for both performance and power and describe several design techniques utilized to accelerate feature extraction and description algorithms on low-cost Zynq SoC FPGAs.**

I. Introduction

In the past decade, computer vision applications have gained significant popularity in their use for mobile, battery powered devices. These devices range from everyday smartphones to autonomously navigating unmanned aerial vehicles (UAVs). While the image processing required by these applications may be transferred to the cloud or other off-device computing engines, because of real-time computing requirements and limited data transfer capabilities, it is desirable for computation to be handled locally whenever possible. However, local computation can be quite challenging for mobile and embedded systems due to the highly computationally intensive nature of computer vision algorithms. In addition, limited size, weight, and battery lifetime of these systems impose further constraints.

Feature detection and feature description are key building blocks of many computer vision algorithms, including image retrieval, biometric identification, and visual odometry (used in the autonomous navigation of robots). A computationally efficient means of detection and analysis of image features is a critical first step in the development of energy-efficient, single-chip solutions for these applications.

In this paper, we present a comparative study of feature detection and description algorithms across a range of embedded platforms. We evaluate these algorithms in terms of run-time performance, power dissipation and energy consumption. In particular, we compare embedded CPU-based, GPU-accelerated, and FPGA-accelerated embedded platforms and explore the implications of various architectural features of these platforms on the execution time and power consumption of these fundamental computer vision algorithms.

In our analysis, we compare off-the-shelf state-of-the-art implementations of feature detection and description algorithms run on both embedded CPUs and GPUs with customized versions run on an embedded FPGA platform. In particular, our FPGA version includes the following optimizations: *i*) a pre-processing stage to eliminate unnecessary computation, *ii*) a feature detection kernel that reduces required memory bandwidth though the use of a zig-zag memory access pattern for image masks, *iii*) a fine-grained power management methodology which turns off downstream hardware when a pixel is guaranteed not to be detected, and *iv*) a feature description kernel pipelined with feature detection. These optimizations result in an FPGA-based platform with increased efficiencies in performance, power dissipation and energy consumption compared to other CPU and GPU embedded platforms.

In addition, we analyze the bottlenecks of our GPU implementation using instruction count evaluation and profiling execution stalls and discuss the design techniques applied to our FPGA design to show how these bottlenecks can be overcome to obtain the high-throughput solutions and hardware-specific power reductions. We conclude that, due to its extra customization and flexibility, our FPGA-accelerated implementation is a promising way forward for the development of low-power, energy-efficient platforms capable of providing real-time performance for complex computer vision based applications such as autonomous navigation, which require the simultaneous execution of multiple kernels. In summary, this work makes the following primary contributions:

- We provide a comprehensive comparison between embedded CPU, GPU and FPGA implementations of feature detection and description algorithms, evaluating their power and performance trade-offs while exploring the architectural advantages and limitations of the acceleration platforms.

- We show that, while the SIMD programming paradigm of GPUs can rival FPGAs for high performance real-time execution of single image processing kernels, GPUs become outclassed by FPGAs over multiple sequential kernels due to a lack of fine grained MIMD programming capabilities.
- We propose a streamlined FPGA implementation of feature detection and description that takes advantage of the highly customizable logic fabric to realize significant improvements in both run-time and power dissipation compared to other embedded solutions.

The rest of the paper is organized as follows: Section II presents an overview of feature detection and description and surveys related work. We also discuss the three different hardware architectures in this study and their ability to exploit parallelism in computer vision algorithms, and give an overview of performance analysis. The details of our feature detector and descriptor algorithms of interest are presented in Section III. Section IV details our software and hardware implementations, while our experimental results and a comparative analysis of the different platform implementations are given in Section V. Finally, we conclude in Section VI with a summary of our findings and a discussion of future work.

II. Background

Feature detection and description algorithms form the basis for the majority of present-day computation-intensive computer vision applications such as 3D mapping, object detection and tracking, and motion and camera pose estimation. This section provides an overview of three versions of these algorithms: Features from Accelerated Segment Test (FAST), FAST + Binary Robust Independent Elementary Features (BRIEF) and FAST + Binary Robust Invariant Scalable Keypoints (BRISK). The section also includes a discussion of the algorithms' amenability for hardware acceleration on three different low-power embedded systems (ARM CPU, GPU and FPGA), and an overview of the metrics utilized to characterize their performance.

A. Overview of Feature Detection

Feature *detection* is a low-level processing operation for identifying pixels of interest in an image which correspond to some elements of a scene that can be reliably located in different views of the same scene. Corners, and edges are typical examples of features.

Previous work on feature detection algorithms primarily involve studies which attempt to detect the highest number of valid features with the least amount of computational effort. The Scale-Invariant Feature Transform (SIFT) algorithm [1] is the most prominent algorithm used for feature detection and has been used as a baseline for most feature detection algorithms since it was first published over 10 years ago. As a more efficient alternative to SIFT, both in terms of speed and computational complexity, the FAST algorithm was proposed [2]. FAST uses corner-based feature detection, as opposed to the Difference of Gaussian (DoG) approach used by SIFT and its faster successor, the Speeded Up Robust Features (SURF) [3] algorithm. Analysis done by Canclini *et al.* on low complexity feature detectors demonstrates definitively the strength of corner based feature detectors over DoG based detectors [4].

There has been a significant amount of research into hardware acceleration of feature detection algorithms. For instance, Bouris *et al.* [5] and Svab *et al.* [6] both presented FPGA implementations for the SURF algorithm. Both works emphasize the power efficiency gained over GPU implementations; however they fall short in terms of comparing FPGA run times to GPU versions. Indeed, GPU implementations in literature, such as [7] and [8], show the performance potential of feature detection working on high-end GPUs. However, these works do not discuss the power and energy repercussions of using such GPUs, which dissipate power in the range of 200W, and the challenges of running these algorithms on tightly constrained embedded platforms. We emphasize that for embedded platforms, both runtime performance and power must be optimized.

On the other hand, a fully customized ASIC implementation of SURF on a $28nm$ CMOS was recently presented [9], where a very low power dissipation of $2.8mW$ was reported. Despite the high computation requirement of the SURF algorithm due to the use of the DoG approach, this work demonstrates the gains that can be achieved by hardware-oriented design optimizations.

There exist other examples in the literature that suggest the use of FPGAs for feature detection applications [10]–[12]. While these works offer promising results, they mainly focus on individual algorithms and fail to compare their results across the board with other embedded system implementations.

B. Overview of Feature Description

Feature extraction involves computing a unique and identifying *descriptor* from the pixels in the region around each point of interest. Descriptors are used to uniquely identify each feature and match regions of features between two or more images.

The SIFT and SURF algorithms also generate descriptors for the features they detect. These descriptors are represented by Histograms of Gradients (HoG). As more efficient alternatives, algorithms such as BRIEF [13] and BRISK [14] use binary feature descriptors, which have the advantage of significantly decreasing the complexity of feature matching by allowing the use of Hamming Distance to measure similarity between two descriptors.

A significant effort has been made to extract and show the potential of such algorithms working on high-end GPUs [7], [8], [15], however these works put no emphasis on the challenges of running these algorithms on tightly constrained embedded platforms, nor mention the repercussion of using such GPUs on power and energy consumption.

In [16], the FAST feature detector is extended by adding the rotational BRIEF feature descriptor to form the Oriented FAST Rotated BRIEF (ORB) feature detection and description algorithm. Lee *et al.* [17] and Kulkarni *et al.* [18] each presented complete hardware implementations for ORB, which emphasized run-time performance gains over their hardware accelerated SIFT and SURF counterparts. They do not, however, provide any analysis of power dissipation.

The work done by Fularz et al. in [19] goes a bit further and implements a hardware accelerated architecture for real-time image feature detection (FAST), description (BRIEF) and matching (Hamming distance) on an FPGA. Their performance measurements, however, are limited to runtime in terms of frames per second (fps), and resource utilization (LUTs, FFs, and BRAMS) in one embedded environment with no power dissipation analysis.

C. Hardware Acceleration of Kernels

Different computational architectures present different opportunities to accelerate feature detection and descriptor generation. In general, computer vision kernels are highly data parallel and map well to streaming architectures. This is particularly true of feature detectors where every pixel is examined in order to determine if it meets the criteria of a feature. These detectors tend to have few data dependencies, limited branching, and few control dependencies. As noted in [20], kernels with these characteristics are strong candidates for GPU acceleration.

Feature descriptor generation, however, requires computation across non-contiguous image sub-regions. This results in more irregular memory access patterns compared to feature detectors. In addition, feature description algorithms tend to have higher complexity and require pre-computation with less data parallelism, but which can benefit from design techniques such as pipelining. This class of algorithms is a stronger candidate for acceleration via FPGAs as compared to GPUs. This distinction will become more apparent in Section V where our FPGA implementation effectively hides the latency for feature description by pipelining with feature detection.

D. Overiew of Performance Analysis

As mentioned in the introduction, a main goal of our work is to better understand the run-time and power dissipation trade-offs of running various implementations of feature detection and descriptor algorithms on different embedded platforms. While prior work [19], [20], [21], and [22] have provided us with some initial understanding on how to evaluate performance metrics on embedded systems, we aim to offer a more comprehensive analysis, again, emphasizing both performance and power. We have chosen to base our analysis on the FAST corner-based feature detector algorithm due to the reduced computation complexity it offers along with comparable performance with respect to the SIFT algorithm. For feature descriptor algorithms, we have chosen to use the BRIEF and BRISK, which use binary descriptors, since they have different use-cases for low-power embedded systems depending on the need for rotation invariance. In addition, BRIEF and BRISK offer up to 2 orders of magnitude faster run-times compared with the HoG-based descriptors SIFT and SURF with comparable precision/recall rates (as will be discussed in the next section).

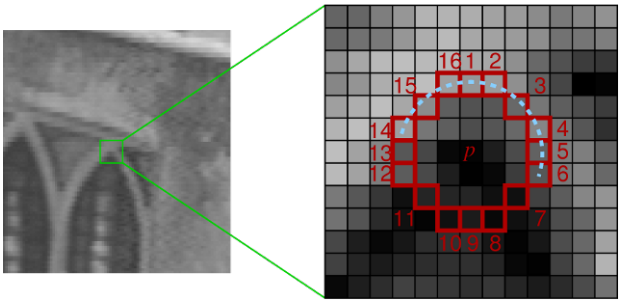

Fig. 1. The Bresenham circle is used to determine if interest point p is a corner feature. Figures taken from [2].

In the next section we review the details of the FAST corner-based feature detection algorithm, as well as the framework for the generation of binary feature descriptors as used in the BRIEF and BRISK algorithms.

III. Feature Detection and Description Details

Corner based feature detectors are derived from the idea of finding rapid changes in direction on image edges to determine a unique image region of interest. In order to identify whether a pixel p with an intensity value I_p is a corner, the FAST detector analyzes a 16 pixel Bresenham circle surrounding p. The Bresenham circle is an approximation of a circle around the center pixel, as shown in Figure 1. A positive detection is declared if n points of this circle form a contiguous segment, which is either darker or brighter than the center pixel, within a pre-defined threshold T. Algorithm 1 shows the pseudo code for FAST.

Algorithm 1 FAST Feature Detection.

1: For each pixel p in an image, assume the intensity of the pixel to be I_p
2: Define a threshold intensity value T
3: Define a 16 pixels Bresenham circle of radius 3 centered around p, where each pixel corresponds to $I_1, I_2, ..., I_{16}$
4: **for** each $i \in I_i$ **do**
5: **if** $I_{i...i+12} + T < por I_{i...i+12} - T > p$ **then**
6: p is a corner
7: **end if**
8: **end for**

Feature descriptors based on Histogram of Gradients (HoG) such as SIFT and SURF require computing the gradient of the image in the region of each feature, which is a very costly process. As shown in Figure 2, SIFT feature descriptors perform best out of commonly used feature descriptors in terms of precision and recall rates, where

precision is the number of relevant detected features over the total number of detected features and recall is the number of relevant detected features over the total number of relevant features. However, when comparing the run time of HoG-based and binary feature descriptors, we observe that SIFT (as a HoG-based descriptor) has a computation time 2 orders of magnitude greater than that of binary descriptors ($540ms$ vs. $3.5ms$ for BRISK, the slowest binary descriptor). The SURF algorithm, which is also HoG-based, improves computation time through the use of integral images; however, it is still an order of magnitude slower than the binary descriptor algorithms which is insufficient for real time use.

Binary descriptors are composed of 3 main sections: a sampling pattern, orientation compensation, and sampling pairs. A region centered around a detected corner needs to be described as a binary string. Given a sampling pattern, pick N pairs of points on the pattern and determine whether the first element or the second element of the pair is greater than the other and define the pair as binary 1 or 0 correspondingly. The resulting N bit vector is the feature descriptor for the said point to be used for feature matching.

The flowchart of the feature detection and description framework is given in Figure 3. The program starts by reading the input frame from the memory. The FAST feature detection algorithm is then applied over each pixel (p) of the frame as a sliding window operation. Despite the non-standard mask of the FAST algorithm, the Bresenham circle, where immediate neighbors of the center pixel are not used for computation, spatial locality can still be used for parallelization of the algorithm. The high spatial locality nature of sliding window operations lend themselves to better parallelization since a pixel is more likely to be accessed when its neighboring pixel has already been accessed; therefore, each memory read for a group of pixels is more likely to access multiple data for immediate use. Both instruction-level and thread-level parallelization techniques can be applied over the algorithm. The pixels over the Bresenham circle are transferred into 16 element arrays where instruction-level parallelism is utilized to compare the values over the circle with the center pixel. Once

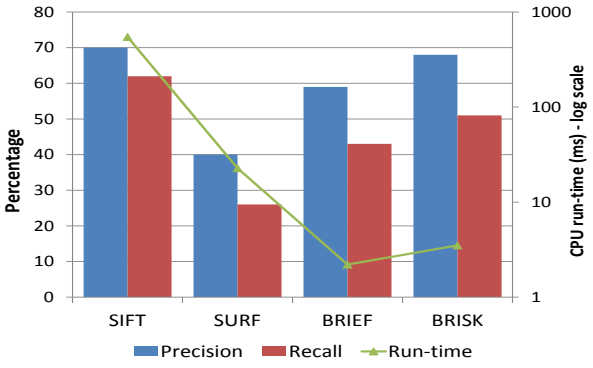

Fig. 2. The precision/recall rate and the run-time comparison of feature descriptors on an Intel i7 CPU.

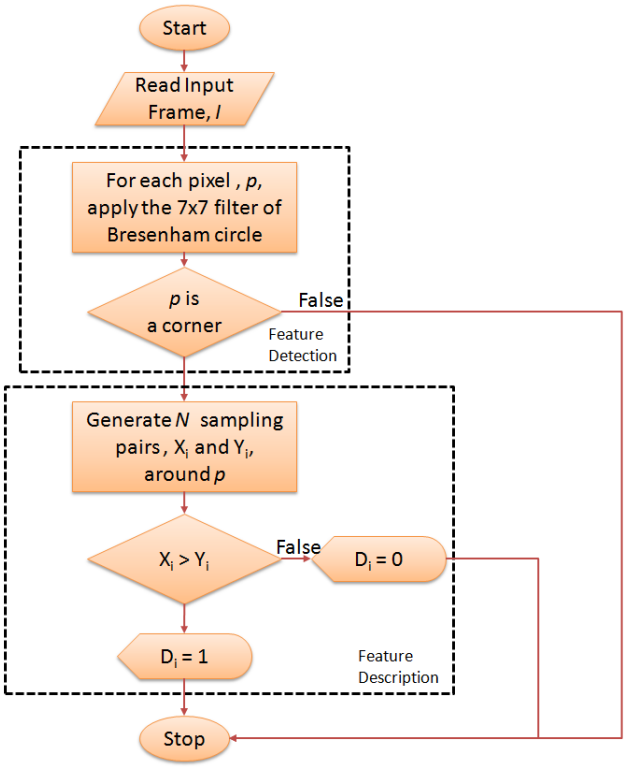

Fig. 3. Flowchart for feature detection and description.

the comparison binary array is generated, existence of a continuous string of 12 bits is checked. If this check returns true, then the center pixel is declared a corner feature. This process is repeated for each pixel of the input frame and can be parallelized over multiple computation units or threads. Once a certain p is found to be a corner, N sampling pairs X_i and Y_i ($\forall i \in N$) surrounding p are evaluated to describe the feature using an N bit descriptor vector D. Each bit of vector D is assigned true (1) or false (0) based on evaluation of $X_i > Y_i$.

IV. Platform Implementations

We have targeted three distinct low-power embedded platforms for evaluating our implementations. For the FPGA design we use a MicroZED development board featuring a 28nm Zynq 7020 SoC, which integrates an Artix-7 FPGA with a dual-core ARM Cortex A9 CPU, and a 1GB DDR3. This platform is logically divided into a *Processing System* (PS) side containing the ARM CPUs and the *Programmable Logic* (PL) side with the FPGA and associated support logic. The DDR3 in the FPGA is used to store input image data, output coordinates, and feature descriptor vectors. A 32-bit AXI Central Interconnect module is used to interface our custom IP module with the DDR3 via the ARM AMBA AXI4-Lite protocol standard at an effective frequency of 111MHz.

For the low-power GPU and embedded CPU platforms, we used the Jetson TK1 development kit. The Jetson board has a 28nm Tegra K1 SoC with an integrated Kepler GPU with 192 CUDA cores that run at 950MHz and a quad-core ARM Cortex A15 CPU that runs at 2.5GHz. The system

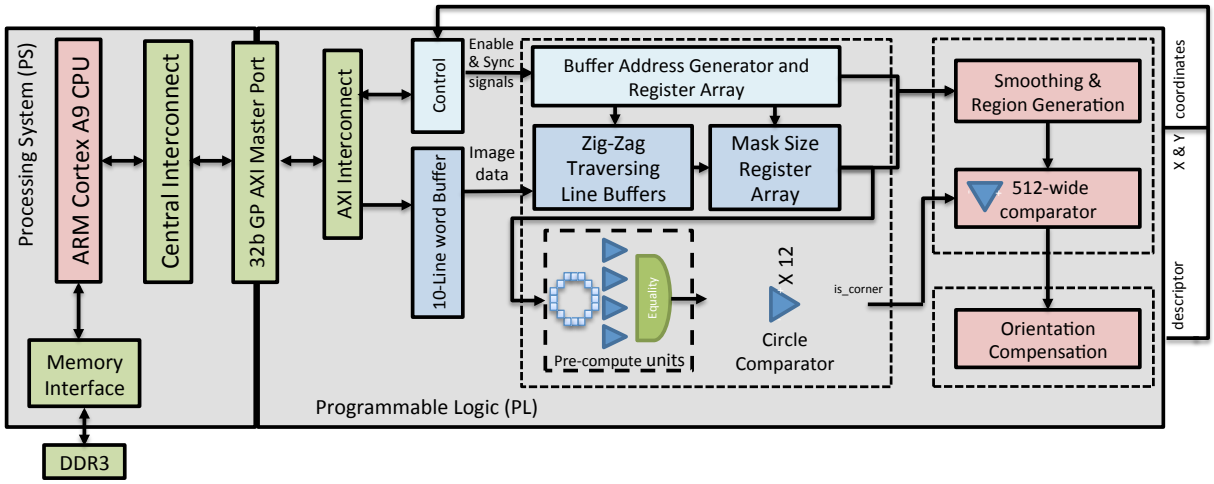

Fig. 4. Top-level block diagram for FPGA implementation with FAST feature detection and BRIEF/BRISK feature description.

has 2GB on board memory. Both the Zynq and Tegra SoCs utilize the same critical feature size (28nm), thus making the architectures the primary differentiator in our experimental evaluation.

The Zynq FPGA uses a bare-metal configuration, and one of two available ARM Cortex-A9 CPUs was used for debugging and initialization purposes. The Zynq FPGA monitors the address space of the DDR3 via the Memory Interface to read and verify its contents while our design is running. Our custom IP module and AXI interconnect module are located on the PL side. The memory interface which directly connects the ARM CPUs to the DDR3 is located in the PS side. The design was simulated and implemented on Vivado 2015.2 IDE and run on the Zynq MicroZED board using Xilinx SDK. The GPU and embedded CPU implementations were tested using the Ubuntu Linux for Tegra distribution and OpenCV version 2.4.10. All of our implementations were tested using an 800×480 Wide VGA (WGA) image resolution which is commonly used for high-quality handheld devices and CMOS image sensors used in robotics.

A. FPGA Architecture

The block diagram of the FPGA hardware is given in Figure 4. We rely on the AXI4-Lite protocol to transfer data between our custom IP and the DDR3 where our image input data and output memory reside. Depending on the algorithm under evaluation, the custom IP module contains the Verilog implementation of FAST, the integrated versions of FAST+BRIEF, or the integrated versions of FAST+BRISK. In our design, the Zynq processing system has the dual role of initiating the main system clock and monitoring the contents of the DDR3. The custom IP module behaves as a master and initiates memory mapped reads and writes to the DDR3, which behaves as a slave in accordance to the AXI4-Lite protocol.

To initiate a memory read, our custom IP waits for the DDR3 to assert a ready signal. This signal is not asserted every clock cycle; hence, to guarantee a continuous flow of input data from the DDR3 to the custom IP, our block diagram, shown in Figure 4, uses a 10-line word buffer to deliver buffered pixel data at a rate of 1 byte per clock cycle as a continuous stream input to FAST, FAST+BRIEF or FAST+BRISK. The output coordinates and/or descriptor vectors produced by the algorithms are then written to the DDR3 one word at a time.

FAST feature detection uses the Bresenham circle mask to traverse an image frame. Even though each mask operation uses 16 pixels, the whole mask size is 7×7, and the whole mask region must be provided to the FAST feature detection computational logic to sustain a single pixel per cycle throughput, requiring a high memory bandwidth. However, on an FPGA implementation, available logic elements can be freely traded for additional storage or computation, allowing us to create line buffers to reuse overlapping pixels for subsequent masks and effectively increasing the bandwidth of our computational logic.

Each of the line buffers are used to store the contents of a single row of the input image data using address accessible 1-D register arrays. For a mask size of $N \times N$, N line buffers are utilized. Each subsequent row of the input image overwrites the contents of the oldest line buffer.

The image corresponding to the 7×7 mask size of the Bresenham circle is stored in a 7×7 register array made up of shift registers. With each pixel read from the input image, the bottom row of the register array is updated with the new data, shifting all the other pixel values horizontally. Meanwhile all other rows of the register array are updated reading the corresponding pixel value from the corresponding line buffer. The matching of the line buffers with the row of the register array is done by utilizing pointers that keep shifting whenever a new line buffer is being written. This architecture enables us to reduce the memory bandwidth requirement of the FPGA design to one pixel per cycle despite the size of the computational mask.

For the reading of the line buffers, we have applied a zigzag access pattern. This approach allows continuous

computation of the mask by utilizing the overlap of pixels in a given image block even through row changes. The filtering starts from the top left pixel location of the input I and proceeds to the following pixel on the same row until the last possible location of this row is filtered. Then, the filter operation continues with the last pixel location of the next row and proceeds with the previous pixel until the first search location of this new row is parsed. Only as many pixels as the width of the filter are required by the computation units in each cycle to calculate the result of filtering regardless of its position in the image. This zigzag flow is enabled by the use of computation units that are capable of shifting data up, left and right, and also by the use of temporary registers that store the values for an up shift of the image block.

For each valid state of the 7×7 register array, the pixels that correspond to the Bresenham circle around the center pixel are evaluated. An initial comparison of 4 pixels intersecting the axis of the circle (labeled as 1, 5, 9 and 13 in Figure 1) is performed to cut down the total number of comparisons, as described in [23]. If this pre-computation returns false, then there is no need to further compare the remaining 12 pixel points of the Bresenham circle since there is no chance of obtaining a continuous 12 pixel ring around the center. The reduction in the number of comparisons does not impact the delay of the computation since the pixels do need to be registered in either scenario to sustain constant throughput, however the reduction of redundant comparison operations and bit propagation reduces the dynamic power dissipation of the circuit.

The feature description in our system has been implemented and evaluated with two different feature descriptor algorithms, BRIEF and BRISK, in order to observe sensitivity to rotation invariance. Both algorithms share the same basic framework presented in Figure 3. A region centered around a detected corner p needs to be described as a binary string. Given a sampling pattern, the feature descriptor generates N sampling pairs for each detected feature and determines whether the first or the second element of each pair is greater than the other and defines the pair as binary 1 or 0 correspondingly. The resulting N bit vector D is the feature descriptor for the said point to be used for feature matching. Here N is equal to 512 for all of our descriptor implementations.

The BRIEF and BRISK descriptor algorithms are implemented as additional pipeline stages in the FAST feature detection implementation. The line buffers used in FAST are incremented in number to cover 31 rows of input image data and are utilized for both the detection and description phases. Once the current pixel coordinate is computed to be a feature, pairing samples are generated from a 31 × 31 sampling window around its location, where the sampling window is stored in a 31×31 register array and traversed in a zig-zag manner as in the case of FAST.

The sampling pairs for both BRIEF and BRISK implementations are fixed according to the algorithm descriptions. Therefore the pairs can be compared to generate the 512-bit binary feature descriptor vector directly from the 31×31 register array. This register array needs to be updated with each new data read, however feature description computation only takes place after a pixel coordinate is already computed as a feature coordinate.

Unlike BRIEF, the BRISK algorithm is an orientation invariant feature descriptor and estimates the orientation of detected features from the selected sampling pairs and rotates the sampling pattern to neutralize the effect of rotation. The sampling points are distinguished as short pairs and long pairs based on their distance to each other, where long pairs are used to determine orientation and short pairs are used for the intensity comparisons that build the descriptor. For BRIEF, a preliminary smoothing operation is applied over the whole image I before the sampling pairs are chosen and compared. The smoothing is applied over the 31×31 sampling window for each detected feature. For BRISK, a simplified Gaussian smoothing is applied for each detected feature coordinate p prior to comparison of sampling pairs and orientation calculation.

B. GPU Architecture

The Tegra K1 SoC combines four ARM Cortex A15 CPU cores with a Kepler class GPU with 192 CUDA cores on the same die. The Kepler GPU has a separate 64kB on-chip L1 cache and a 128kB on-chip L2 cache, while the CPUs and GPU share 2GB of off-chip memory. GPU programming with the Tegra K1 is dominated by two factors common to all GPUs. The first is the highly parallel SIMD nature of GPU programming. The second is the unique memory model.

The highly parallel nature of GPU programming makes large amounts of branches and control logic expensive to implement. GPU threads on the K1 are organized into blocks that can access global GPU memory or a memory space shared only within the thread block. Shared memory on the K1 is implemented with the 64kB L1 cache so most memory accesses for the kernels under study will be to the 2GB off-chip global memory space.

We implement our GPU kernels using version 2.2.10 of the OpenCV computer vision library and NVIDIA's CUDA API. Both the CPU and GPU OpenCV library for feature detection and description are publicly available and highly optimized by the community to be used for performance critical applications. The FAST feature detector kernel can be mapped almost entirely to the Kepler GPU. This kernel contains a modest amount of branches and no loops. However, the feature descriptor implementations of BRISK, and BRIEF have large sections of initialization code which does not map well to a GPU architecture. These code sections are executed on a single ARM core, limiting the achievable speedup. It is possible that some initialization code can be mapped to other CPU cores, thus achieving a small additional speedup. However, the additional speedup was judged to be too modest to warrant inclusion in our analysis.

V. Experimental Results

The run-time, power, and energy comparisons of the FAST feature detector with and without the BRIEF/BRISK feature description algorithms on various embedded systems is given in Figure 6. All implementations were verified against the baseline OpenCV CPU implementation of the algorithms for accuracy. We have used the Jetson TK1 development kit for the Embedded CPU and GPU implementations. The FPGA implementation results were taken from the MicroZED development board running at 111 MHz. For all platforms, we have measured the power by intercepting the current between the power supply and the system by a 1 mΩ shunt resistor. The voltage across the resistor is measured using a multi-meter to calculate the total power of each system. By measuring the total power of our embedded system boards, we are able to compare the power cost of using each platform as part of a complete system, such as one used to control UAVs.

As a sliding window operation, FAST has a highly predictable data access pattern traversing each input image frame. On the other hand, feature description algorithms traverse over a list of detected features and accesses the sampling window around each feature coordinate, resulting in an irregular access pattern. This irregularity impacts the run-time of the CPU the most, whereas the GPU architecture can handle irregular accesses to a 2-D data relatively well. For the FPGA implementation, the flexibility to trade-off resources for performance allows us to fully pipeline detection and description computation, completely eliminating the need for additional memory accesses and conserving the same throughput and thus very similar run-times. With a frame rate of 36 fps, our GPU implementation for FAST+BRISK is within real-time requirements. However, it cannot keep up with the maximum capturing rate of 60 fps available in most modern CMOS image sensors used on UAVs. Our FPGA implementation for FAST+BRISK can reach up to 147 fps, which is well within real-time requirements and also has the potential to process image resolutions larger then WGA in acceptable frame rates.

We have used NVidia Visual Profiler to identify the performance bottlenecks for our GPU implementations and analyze the underlying reasons for execution stalls. Figure 5 displays the source of execution stalls for both our FAST and BRIEF implementations. The dominant causes of execution stalls include Execution Dependency, Pipe Busy, Memory Throttle and Not Selected. Execution dependency stalls occur when an instruction is waiting for at least one of its inputs to be computed by earlier instructions. Pipe busy stalls are observed when the computation unit required for the instruction is not available. Memory throttle stalls are due to requiring a larger number of memory requests then the capabilities of the load/store unit and thus not being able to accommodate all the requests in time. Not selected stalls occur when the warp scheduler gives priority to another kernel over the computation kernel and thus not allocating the required resources.

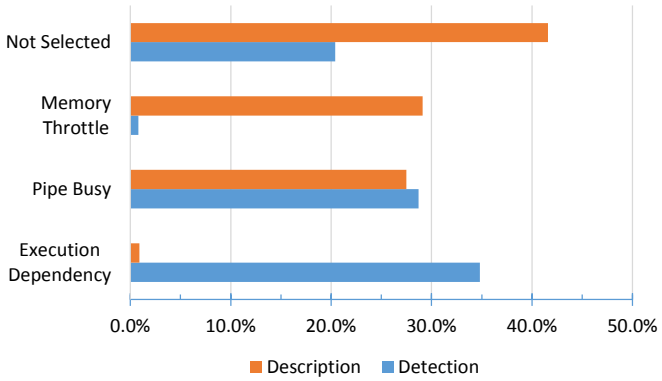

Fig. 5. Issue Stall Reasons for FAST and BRIEF implementations on GPU

We observe that the feature detection algorithm is heavily stalled due to execution dependencies compared to the description computation. On the other hand the description computation is heavily stalled due to memory throttle. This shows us that the feature detection algorithm could benefit greatly from further instruction level parallelism whereas the feature descriptor algorithm would benefit from better data management.

The total instruction count for our GPU and FPGA implementations are given in Table I. The GPU instruction counts were taken from NVIDIA visual profiler. The corresponding instruction counts for the FPGA were estimated from the design architecture and system level simulation. We estimate the Load/Store instruction count as the number of accesses to the DDR3 and disregard the accesses to the internal register buffers. The number of integer instructions is estimated based on the total number of cycle operations the architecture takes, when all the pipeline stages are unrolled. Therefore, each cycle of operation a single feature candidate or descriptor goes through is evaluated as an instruction. There can be multiple instructions in different pipeline layers, giving an instruction count metric comparable to that of a CPU or GPU. The control instructions were estimated via a code level analysis of branch instructions. Similar to the total number of integer instructions, the estimations are made by multiplying total number of branches per mask operation with the total number of mask operations applied over an image and thus each pipeline stage is considered as a separate MIMD instruction.

We observe that the feature detection kernel (i.e., FAST) corresponds to the majority of the floating point and integer instructions, making up over 70% of the total instructions issued. On the other hand, the number of load/store instructions are significantly higher for the feature description kernel (i.e., BRIEF) specifically for the GPU. For the FPGA implementation, the image is read from the external memory only once for both detection and description computations into our line buffers, drastically reducing the number of memory accesses. Thereafter the corresponding pixel data is propagated through our computation logic

TABLE I
INSTRUCTION NUMBER COMPARISON BETWEEN GPU AND FPGA IMPLEMENTATIONS

	Load/Store	Floating Point and Integer
FAST FPGA	384000	2688000
FAST+BRIEF FPGA	384000	3456000
FAST GPU	304629	46310713
FAST+BRIEF GPU	2560475	64262713

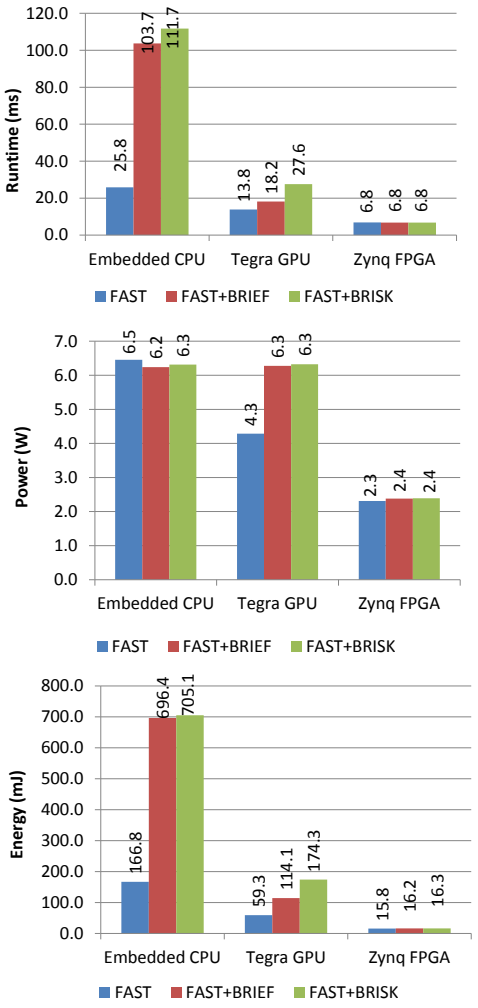

Fig. 6. Run-time and power results for FAST, BRIEF and BRISK algorithms over various embedded systems.

until it is discarded.

As seen in Figure 6, the power consumption of feature description algorithms are very similar for our embedded CPU and GPU implementations whereas the feature detection consumes significantly less power on the GPU. The sliding window operation for the detection part is efficiently distributed to the low power cores of the GPU whereas the feature description suffers from the bottleneck of irregular memory accesses in terms of power consumption.

The resource utilization of our FPGA implementations are provided in Table II. The additional logic for description computation for the FPGA implementation is readily

TABLE II
RESOURCE UTILIZATION ON THE ZYNQ FPGA

	LUT	FFs	BRAMs
FAST	4564	1551	8
FAST+BRIEF	14398	2093	11
FAST+BRISK	25575	7115	11

available to compute the description at each pixel coordinate, however unnecessary bit propagation is eliminated by registering the inputs in order to minimize the cost of extra circuitry on power.

The run-time performance for the GPU and FPGA are more comparable. However the highly customizable architecture of the FPGA lends itself much better to optimization of FAST+BRIEF and FAST+BRISK implementations. When detection is pipelined with description our results show that the GPU lags behind in performance due to a lack of MIMD capabilities. Combined with the lower power dissipation overhead for the FPGA boards, we can see a clear advantage of FPGAs over the other platforms in terms of energy consumption with measurements of 705mJ, 174mJ, and 16mJ for feature detection and description of WGA size frames on embedded CPUS, GPUs and FPGAs respectively, giving the FPGA a 98% advantage over the CPU implementation and a 90% advantage over the GPU implementation.

VI. CONCLUSION

In this paper we presented a comparative analysis of the FAST feature detection algorithm along with BRIEF and BRISK feature description algorithms on various embedded systems (embedded CPUs, GPUs and FPGAs) in terms of run-time performance, power, and energy. We determined that the utilization of hardware-oriented and power-aware design decisions such as deep pipelining, continuous filter flow, and pre-computation steps allow high-throughput FPGA implementations to outperform state-of-the-art embedded CPUs and GPUs in terms of both power and performance. We show that despite the high-level parallelization GPUs provide, computation of multiple kernels are highly bounded by kernel scheduler and memory bottlenecks whereas customization of FPGAs on layers can tackle operation of multiple kernels much more efficiently. We have shown that the initial profiling of GPU implementations can allow the designers to identify bottlenecks in a design and deduce whether these bottlenecks can yield performance gains with custom hardware programmability of FPGAs. This analysis constitutes a first step toward high performance computer vision based embedded systems. Future work will build upon these implementations and results by integrating additional hardware accelerated kernels such as simultaneous mapping and localization (SLAM) necessary for autonomous navigation of UAVs.

VII. ACKNOWLEDGMENTS

This work was supported in part by NASA grant number NNX13AN07A.

REFERENCES

[1] D. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004. [Online]. Available: http://dx.doi.org/10.1023/B%3AVISI.0000029664.99615.94

[2] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *Proceedings of the 9th European Conference on Computer Vision - Volume Part I*, ser. ECCV'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 430–443. [Online]. Available: http://dx.doi.org/10.1007/11744023_34

[3] H. Bay, T. Tuytelaars, and L. Van Gool, "SURF: Speeded up robust features," in *Computer vision–ECCV*. Springer, 2006, pp. 404–417.

[4] A. Canclini, M. Cesana, A. Redondi, M. Tagliasacchi, J. Ascenso, and R. Cilla, "Evaluation of low-complexity visual feature detectors and descriptors," in *Digital Signal Processing (DSP), 2013 18th International Conference on*, July 2013, pp. 1–7.

[5] D. Bouris, A. Nikitakis, and I. Papaefstathiou, "Fast and efficient FPGA-based feature detection employing the SURF algorithm," in *Field-Programmable Custom Computing Machines (FCCM)*, May 2010, pp. 3–10.

[6] J. Svab, T. Krajnik, J. Faigl, and L. Preucil, "FPGA based speeded up robust features," in *Technologies for Practical Robot Applications, 2009. TePRA 2009. IEEE International Conference on*, Nov 2009, pp. 35–41.

[7] H. Xie, K. Gao, Y. Zhang, J. Li, and Y. Liu, "GPU-based fast scale invariant interest point detector," in *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on*, March 2010, pp. 2494–2497.

[8] R. Phull, P. Mainali, Q. Yang, P. R. Alface, and H. Sips, "Low complexity corner detector using CUDA for multimedia applications," *MMEDIA 2011*, 2011.

[9] D. Jeon, M. Henry, Y. Kim, I. Lee, Z. Zhang, D. Blaauw, and D. Sylvester, "An energy efficient full-frame feature extraction accelerator with shift-latch FIFO in 28 nm CMOS," *Solid-State Circuits, IEEE Journal of*, vol. 49, no. 5, pp. 1271–1284, May 2014.

[10] M. Schaeferling and G. Kiefer, "Object recognition on a chip: A complete surf-based system on a single FPGA," in *Reconfigurable Computing and FPGAs (ReConFig), 2011 International Conference on*, Nov 2011, pp. 49–54.

[11] G. Van der Wal, D. Zhang, I. Kandaswamy, J. Marakowitz, K. Kaighn, J. Zhang, and S. Chai, "FPGA acceleration for feature based processing applications," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2015.

[12] H. Chang, I. Jiang, H. Hofstee, D. Jamsek, and G. Nam, "Feature detection for image analytics via FPGA acceleration," *IBM Journal of Research and Development*, vol. 59, no. 2/3, pp. 8:1–8:10, March 2015.

[13] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "BRIEF: Binary robust independent elementary features," *Computer Vision–ECCV 2010*, pp. 778–792, 2010.

[14] S. Leutenegger, M. Chli, and R. Y. Siegwart, "BRISK: Binary robust invariant scalable keypoints," in *Computer Vision (ICCV), 2011 IEEE International Conference on*. IEEE, 2011, pp. 2548–2555.

[15] M. Schweitzer and H.-J. Wuensche, "Efficient keypoint matching for robot vision using GPUs," in *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*, Sept 2009, pp. 808–815.

[16] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: An efficient alternative to SIFT or SURF," in *Proceedings of the 2011 International Conference on Computer Vision*, ser. ICCV '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 2564–2571. [Online]. Available: http://dx.doi.org/10.1109/ICCV.2011.6126544

[17] K.-Y. Lee, "A design of an optimized ORB accelerator for real-time feature detection." *International Journal of Control & Automation*, vol. 7, no. 3, 2014.

[18] A. Kulkarni, J. Jagtap, and V. Harpale, "Object recognition with ORB and its implementation on FPGA," *International Journal of Advanced Computer Research*, vol. 3, no. 3, pp. 164–169, 2013.

[19] M. Fularz, M. Kraft, A. Schmidt, and A. Kasinski, "A high performance FPGA based image feature detector and matcher based on the FAST and BRIEF algorithms," *International Journal of Advanced Robotic Systems*, vol. 12, no. 141, 2015.

[20] S. Che, J. Li, J. Sheaffer, K. Skadron, and J. Lach, "Accelerating compute-intensive applications with GPUs and FPGAs," in *Application Specific Processors, 2008. SASP 2008. Symposium on*, June 2008, pp. 101–107.

[21] S. Asano, T. Maruyama, and Y. Yamaguchi, "Performance comparison of FPGA, GPU and CPU in image processing." in *FPL*, M. Danek, J. Kadlec, and B. E. Nelson, Eds. IEEE, 2009, pp. 126–131. [Online]. Available: http://dblp.uni-trier.de/db/conf/fpl/fpl2009.html#AsanoMY09

[22] J. Fowers, G. Brown, P. Cooke, and G. Stitt, "A performance and energy comparison of FPGAs, GPUs, and multicores for sliding-window applications," in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, ser. FPGA '12. New York, NY, USA: ACM, 2012, pp. 47–56. [Online]. Available: http://doi.acm.org/10.1145/2145694.2145704

[23] E. Rosten and T. Drummond, "Fusing points and lines for high performance tracking," in *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, vol. 2, Oct 2005, pp. 1508–1515.