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

Onur Ulusel, Christopher Picardo, Christopher Harris, Sherief Reda, R. Iris Bahar,
School of Engineering, Brown University
Image Processing in Mobile Systems

• Image processing is everywhere!
  – Input data has changed from words/numbers to images
  – Sensors have improved dramatically

• Image processing is a major driving factor in technological advancement
  – Autonomization relies on image processing

• Mobile/Embedded platforms??
  – Real-time computing + limited data bandwidth ➔ prefer local computing to offloading to cloud
  – BUT image processing can be very computationally intensive and power hungry
Accelerating Image Processing on Low Power Embedded Platforms

• Meeting real time image processing requirements for many of these applications requires HW assisted acceleration

• Which algorithms do we accelerate?
  – *Feature detection* and *feature description* are key building blocks for image retrieval, biometric identification, visual odometry, etc.
  – Computational efficient detection and analysis of image features is critical for *performance* and *energy-efficiency*
Hardware Acceleration for Energy Constrained Image Processing

- Low power embedded platforms
  - Field Programmable Gate Arrays (FPGAs)
  - Graphical Processing Units (GPUs)
  - Low power general processors

<table>
<thead>
<tr>
<th></th>
<th>FPGA</th>
<th>GPUs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xilinx</td>
<td>Virtex 6</td>
<td>1532 core NVIDIA</td>
</tr>
<tr>
<td>Xilinx Zynq</td>
<td>7020</td>
<td>GeForce GTX 680</td>
</tr>
<tr>
<td></td>
<td>192 core NVIDIA</td>
<td>Jetson TK1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Power</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;5W</td>
</tr>
<tr>
<td></td>
<td></td>
<td>195 W</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;12W</td>
</tr>
</tbody>
</table>

Which platform is best?
Our Contributions

• Comparative study of feature detection and description algorithms
  – What are their computation kernel characteristics?

• Comparative study of platforms for embedded applications
  – Advantages/disadvantages of each platform?

• Accelerating algorithms on different platforms
  – How can algorithms be modified to better exploit available hardware of each platform?
  – How does performance compare in terms of run time and energy consumption?
Feature Detection

• **What is a ‘feature’?**
  – An “interesting” part of an image that can be used to identify objects

• **Examples:** Edges, corners, ridges, blobs

Slide adapted from Darya Frolova, Denis Simakov, Weizmann Institute
Feature Description

• Given the features, *uniquely describe* them so they can be matched in other images
• Descriptors summarize characteristics of the features
  – E.g., intensity, orientation
• Descriptors should be distinctive and insensitive to local image deformations.

Accuracy and Run-time Comparisons

- **HoG (Histogram of Gradient) based Descriptors**
  - SIFT: Scale-Invariant Feature Transform
  - SURF: Speeded Up Robust Features

- **Binary Feature Descriptors**
  - BRIEF: Binary Robust Independent Elementary Features
  - BRISK: Binary Robust Invariant Scalable Keypoints
FAST: Features from Accelerated Segment Test

Pre-compare pixels 1, 5, 9, and 13 to determine possibility for continuity.

On average 98.5% of the comparisons fail the continuity test at the pre-compare stage.

Rosten and Drummond, ECCV’06
**BRIEF: Binary Robust Independent Elementary Features**

- **Compare intensities** of pairs of points using Hamming distance

- **BRIEF Sampling pattern**
  - 512 sampling pairs
  - For each pair, $X_i$ is at (0,0) and $Y_i$ takes all possible values from coarse polar grid
  - Sampling pairs are generated from a 31×31 region around center pixel

Chosen sampling pattern results in a 512-bit characterization array
BRISK: Binary Robust Invariant Scalable Keypoints

• BRISK uses custom sampling pattern
• 512 sampling pairs generated from a 31×31 region (like BRIEF)
• Distinguishes between short/long pairs
  – Short pairs used similar to BRIEF to generate descriptor vectors based on intensity comparisons
  – Long pairs used for orientation computation by rotating sampling pattern
For each pixel, \( p \), apply the 7x7 filter of Bresenham circle.

If \( p \) is a corner:

- Generate \( N \) sampling pairs, \( X_i \) and \( Y_i \), around \( p \).

- For each sampled pair, \( X_i > Y_i \):
  - \( D_i = 1 \)
  - \( D_i = 0 \)

- Feature Detection

- Brief Feature Description

- Stop

Algorithm Flowchart

- FAST feature detection + BRIEF feature description
- Obtaining sampling window for feature description requires irregular access pattern
Algorithm Flowchart

- FAST feature detection + BRISK feature description
- BRISK requires an extra step for orientation compensation
  - A significant amount of extra hardware resources for this step
Experimental Embedded Platforms

• **FPGA:** MicroZED development board:
  – 28nm Zynq 7020 SoC
  – Artix-7 FPGA + 1GB DDR3
  – dual-core Arm Cortex A9 CPU (for debug and init. only)

• **GPU & CPU:** Jetson TK1 development kit
  – 28nm Tegra K1 SoC
  – Kepler GPU with 192 CUDA cores @ 950MHz
  – Quadcore ARM Cortex A15 CPU @ 2.5GHz (single core activated)
  – 2GB Memory
  – Running OpenCV versions of FAST, BRIEF, BRISK
Feature Detection & Description: Block Diagram
Feature Detection & Description: 
Block Diagram

FAST Feature Detection
Feature Detection & Description: Block Diagram
Feature Detection & Description: Block Diagram

- ARM Cortex A9 CPU
- Memory Interface
- DDR3
- Central Interconnect
- 32b GP Axi Master Port
- AXI Interconnect
- 10-Line Buffer
- Image data
- Enable & Sync signals
- Pre-compute units
- Orientation Compensation
- BRISK Descriptor

Buffer Address Generator and Register Array
Zig-Zag Traversing Line Buffers
Mask Size Register Array

Smoothing & Region Generation
N-wide comparator

AXI Interconnect
Memory Interface
Processing System (PS)
Programmable Logic (PL)
Feature Detection & Description: Block Diagram

- Feature detection logic
- Descriptor logic
- ARM Cortex A9 CPU
- Central Interconnect
- 32b GP Axi Master Port
- X1 Interconnect
- Enable & Sync
- Image data
- Buffer Address Generator and Register Array
- Zig-Zag Traversing Line Buffers
- Mask Size Register Array
- 30-Line word Buffer
- Pre-compute units
- Circle Comparator
- X 12
- is_circular
- Descriptors logic
- Data control logic for detection and description
- Smoothing & Region Generation
- N-wide comparator
- Orientation Compensation
- DDR3
- Memory Interface
- Processing System (PS)
- Control
- Programmable Logic (PL)
Results: Run-time

- Intel i7 CPU: 4.4 ms, 36.4 ms, 50.8 ms
- ARM on Jetson: 25.8 ms, 103.7 ms, 111.7 ms
- Tegra GPU on Jetson: 13.8 ms, 18.2 ms, 27.6 ms
- Zynq FPGA: 13.7 ms, 13.7 ms, 13.7 ms

Chart showing run-times for different processors and combinations of FAST, FAST+BRIEF, and FAST+BRISK algorithms.
Results: Power & Energy

![Power Consumption Chart](image)

- Intel i7 CPU: 21.5 W, 22.2 W, 22.4 W
- Embedded CPU: 6.5 W, 6.2 W, 6.3 W
- Tegra GPU: 4.3 W, 6.3 W, 6.3 W
- Zynq FPGA: 2.20 W, 2.27 W, 2.31 W

![Energy Consumption Chart](image)

- Intel i7 CPU: 94.0 mJ, 809.4 mJ, 1137.7 mJ
- Embedded CPU: 166.8 mJ, 696.4 mJ, 705.1 mJ
- Tegra GPU: 59.3 mJ, 114.1 mJ, 174.3 mJ
- Zynq FPGA: 30.00 mJ, 30.96 mJ, 31.58 mJ

Legend:
- FAST
- FAST+BRIEF
- FAST+BRISK

(detection)
(detection + description)
Results: Profiling

- Feature description stalled due to memory throttle
  - Needs better data management

- Feature description for GPU implementation has bump in load/store ops
  - Almost 10X more than just FAST
### Results: FPGA Resource Utilization

#### Resource utilization

<table>
<thead>
<tr>
<th></th>
<th>Lookup Tables</th>
<th>Flip Flops</th>
<th>Block RAMs</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAST</td>
<td>4564</td>
<td>1551</td>
<td>8</td>
</tr>
<tr>
<td>FAST + BRIEF</td>
<td>14398</td>
<td>2093</td>
<td>11</td>
</tr>
<tr>
<td>FAST + BRISK</td>
<td>25575</td>
<td>7115</td>
<td>11</td>
</tr>
</tbody>
</table>

#### Distribution of resources

- **BRISK** requires a significant amount of extra resources for smoothing and orienting.
- Extra resources do not translate to much extra power.
Conclusions

• FPGA outperforms CPUs and GPUs in terms of power & performance
  – FAST + BRISK: 36 fps vs. 147 fps
  – FPGA amenable to various HW optimizations:
    • deep pipelining, optimized memory access, pre-computation
• FPGA implementations better for handling multiple kernels
  – For GPUs, multiple kernels highly bounded by kernel scheduler and memory bottlenecks
  – FPGA customization on layers better for tackling operations on multiple kernels.
• Use profiling on GPU implementation as first step to FPGA optimization
  – identify nature of bottlenecks
  – Customized FPGA HW can often better manage certain types of bottlenecks