OpenCV on Zynq:
Accelerating 4k60 Dense Optical Flow and Stereo Vision

Kamran Khan, Product Manager, Software Acceleration and Libraries
July 2017
Agenda

➤ Why Zynq SoCs for Traditional Computer Vision
➤ Automated Flow for OpenCV HW Acceleration
➤ Case Study
OpenCV Needs Acceleration in Embedded

Typical ARM Cortex-A53

<table>
<thead>
<tr>
<th>Typical Requirement</th>
<th>&gt; 30 FPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Harris Corner</td>
<td>2.4 FPS</td>
</tr>
<tr>
<td>Stereo Depth Map</td>
<td>2.1 FPS</td>
</tr>
<tr>
<td>Dense Optical Flow</td>
<td>0.1 FPS</td>
</tr>
</tbody>
</table>

![Graph showing OpenCV needs acceleration in embedded systems.](image)

### OpenCV dominates development.
An astonishing 89% of developers report it as one of their top-three computer vision libraries or APIs. OpenCV 3.0 adds functionality (e.g., deep learning), which is likely responsible for some of its increase in popularity.

Zynq Offers the Most Efficient CV Acceleration
Zynq Offer Superior Performance, Latency

**Real Time Applications Latency**

- **<10 ms latency**
- **42x Frames/sec/watt**

**Computer Vision**

- Xilinx Benchmark
- Xilinx Benchmark

---

### Computer Vision Performance

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Xilinx ZU9</th>
<th>Xilinx ZU5</th>
<th>eGPU*</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CV:: StereoLBM @1080p</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frames/s</td>
<td>700</td>
<td>296</td>
<td>43</td>
</tr>
<tr>
<td>Power (W)</td>
<td>4.8</td>
<td>3.3</td>
<td>7.9</td>
</tr>
<tr>
<td>Frames/s/watt</td>
<td>145.8</td>
<td>89.7</td>
<td>5.4</td>
</tr>
<tr>
<td><strong>CV:: LK Dense Optical Flow @720p</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frames/s</td>
<td>170</td>
<td>73</td>
<td>7</td>
</tr>
<tr>
<td>Power (W)</td>
<td>4.8</td>
<td>3.3</td>
<td>7.9</td>
</tr>
<tr>
<td>Frames/s/watt</td>
<td>35.4</td>
<td>22.1</td>
<td>0.9</td>
</tr>
</tbody>
</table>

- eGPU = nVidia Tegra X1 using VisionWorks for StereoLBM and OpenCV4Tegra for OpticalFlow
- All benchmarks utilize as much resources as possible on GPU (~99%) and programmable logic (~70%)
Why So Good? Efficient Window-based Streaming

Typical SoC

<table>
<thead>
<tr>
<th>DSP/GPU</th>
<th>Optical Flow</th>
<th>Stereo Depth</th>
<th>SfM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Image Sensor

DDR

CPUs

Programmable Logic

<table>
<thead>
<tr>
<th>Optical Flow</th>
<th>Stereo Depth</th>
<th>SfM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Image Sensor

DDR

CPUs
Debunking “Zynq SoC is Hard to Program”

- C/C++/OpenCL Creation
- Profiling to Identify Bottlenecks
- System Optimizing Compiler
- Computer Vision
- Machine Learning
- Optimized Accelerators & Data Motion Network
- Scheduling of Pre-Optimized Neural Network Layers

- DNN
- CNN
- GoogLeNet
- SSD
- FCN...

- .prototxt
- & Trained Weights
OpenCV Support with Automatic HW Acceleration

1. Cross-compile OpenCV application to Zynq (ARM A9/A53)
2. Profile and identify bottleneck functions
3. Minimal changes to the code and set functions to hardware. Compile using SDSoC
4. Run on a Zynq board

```
main()
{
    cv::imread(A);
    cv::stereoRectify(A,B,C,D);
    cv::stereoLBM(C,D,out);
    cv::imshow(out);
}
```

```
main()
{
    cv::imread(A);
    xf::stereoRectify<line>(A,B,C,D);
    xf::stereoLBM<win,n_disp>(C,D,out);
    cv::imshow(out);
}
```
xfOpenCV: HW Accelerated OpenCV Functions

<table>
<thead>
<tr>
<th>Level 1</th>
<th>Level 2</th>
<th>Level 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Absolute difference</td>
<td>Channel combine</td>
<td>Box</td>
</tr>
<tr>
<td>Accumulate</td>
<td>Channel extract</td>
<td>Gaussian</td>
</tr>
<tr>
<td>Accumulate squared</td>
<td>Color convert</td>
<td>Median</td>
</tr>
<tr>
<td>Accumulate weighted</td>
<td>Convert bit depth</td>
<td>Sobel</td>
</tr>
<tr>
<td>Arithmetic addition</td>
<td>Table lookup</td>
<td>Custom convolution</td>
</tr>
<tr>
<td>Arithmetic subtraction</td>
<td>Histogram</td>
<td>Gradient Phase</td>
</tr>
<tr>
<td>Bitwise: AND, OR, XOR, NOT</td>
<td>Gradient Phase</td>
<td>Dilate</td>
</tr>
<tr>
<td>Pixel-wise multiplication</td>
<td>Min/Max Location</td>
<td>Erode</td>
</tr>
<tr>
<td>Integral image</td>
<td>Mean &amp; Standard Deviation</td>
<td>Bilateral</td>
</tr>
<tr>
<td>Gradient Magnitude</td>
<td>Thresholding</td>
<td></td>
</tr>
</tbody>
</table>

© Copyright 2017 Xilinx
Custom CV Function / Library Creation Flow

1. Cross-compile to Zynq (ARM A9/A53)
2. Write custom CV function in C, C++ or OpenCL. Optimize for hardware using HLS
3. Assign functions to hardware. Compile using SDSoC
4. Run on a Zynq board

```cpp
main()
{
    cv::imread(A);
    XF::stereoRectify<line>(A,B,C,D);
    XF::stereoLBM<win,n_disp>(C,D,E);
    CUSTOM.CV(E,out);
    cv::imshow(out);
}
```

```
CUSTOM.CV(E,out){
    #pragma HLS PIPELINE
    for(...){
        #pragma HLS UNROLL
        for(...){ ... }
    }
}
```

<table>
<thead>
<tr>
<th>HW functions</th>
<th>Clock Frequency (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>stereoRectify</td>
<td>300</td>
</tr>
<tr>
<td>stereoLBM</td>
<td>300</td>
</tr>
<tr>
<td>CUSTOM.CV</td>
<td>300</td>
</tr>
</tbody>
</table>

© Copyright 2017 Xilinx
Example: 4K60 LK Dense Optical Flow

```c
main()
{
  imread(A);
  imread(B);
  denseOpticalFlowPyr1tr(A,B,out)
  imshow(out);
}
```

- Xilinx ZU9
  - Frames/s: 60
  - Power (W): 4.8
  - Latency (ms): 16.7
  - Utilization: 15%

- MIPI
- AXI
- SW
- HW
- Linux
- Libraries
- Drivers
- AXI
- HDMI
- DMA
- AXI-S

- nVidia number using CUDA OpenCV
- Both Xilinx and nVidia benchmarks do not include the camera inputs and HDMI/DP
- LK dense optical flow, non-pyramidal, non-iterative, Window size 53x53
Example: Stereo Depth Map

- nVidia number using CUDA OpenCV
- SAD based stereo localBM
- Both Xilinx and nVidia benchmarks do not include the camera inputs and HDMI/DP outputs

<table>
<thead>
<tr>
<th>Xilinx ZU9</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Frames/s</td>
<td>140</td>
</tr>
<tr>
<td>Power (W)</td>
<td>4.8</td>
</tr>
<tr>
<td>Latency (ms)</td>
<td>7.1</td>
</tr>
<tr>
<td>Utilization</td>
<td>14%</td>
</tr>
</tbody>
</table>

```
main(){
imread(A);
imread(B);
stereoRectify(A, B, C, D);
stereoLBM(C, D, out);
imshow(out);
}
```
Step 1: Port Desktop OpenCV Application to Zynq

- Simply import the C/C++ projects with OpenCV APIs into SDSoC
- All necessary OpenCV compile / linking environments for ARM are provided
- Ready-to-compile!
Step 2: Assign Functions to Hardware Acceleration

- Minor mods needed to use OpenCV libraries for hardware acceleration
  - Namespace change: “cv::” to “xF::”
  - Add template parameters for optimized hardware generation

- Simply assign critical functions to hardware

```c
uint16_t width = in_gray.cols;
uint16_t height = in_gray.rows;
xF::Mat<XF_RGBI, HEIGHT, WIDTH, NPC1> _src(height, width);
xF::Mat<XF_RGBI, HEIGHT, WIDTH, NPC1> _dst(height, width);
_src.copyTo(in_gray.data);

xF::BilateralFilter<FILTER_WIDTH, XF_BORDER_REPLICATE, XF_RGBI, HEIGHT, WIDTH, NPC1>(_src, _dst, sigma_space);

out_img.data = _dst.copyFrom();
imwrite("output HLS.png", out_img);
absdiff(ocv_ref, out_img, diff); // Compute absolute difference image
// Find minimum and maximum differences.
```
Step 3: Estimate Performance and Build

- Fast estimation in minutes to get system-level performance and HW utilization
- Build the full system with a click of button

Details

Performance estimates for 'xFBilateralFilter_3_1_0_1080_1 ...'

Hardware accelerated (Estimated cycles) 30185245

Resource utilization estimates for Hardware functions

<table>
<thead>
<tr>
<th>Resource</th>
<th>Used</th>
<th>Total</th>
<th>% Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSP</td>
<td>22</td>
<td>2520</td>
<td>0.87</td>
</tr>
<tr>
<td>BRAM</td>
<td>3</td>
<td>912</td>
<td>0.33</td>
</tr>
<tr>
<td>LUT</td>
<td>7061</td>
<td>274080</td>
<td>2.58</td>
</tr>
<tr>
<td>FF</td>
<td>6627</td>
<td>548160</td>
<td>1.21</td>
</tr>
</tbody>
</table>

ARM Executable

HW bitstream

Linux kernel, Rootfs and Boot files
Step 4: Run on a Board and Collect Traces

- Send command to accelerator to start
- Wait for core to be started
- Timeline units are in seconds
- Start data transfer from PS
- Start data receive from PS side
- Wait for data transfers to complete
- Accelerator active
- Data transfer active
Zynq SoCs offer superior performance and lower latency compared to other SoC offerings

reVISION stack on SDSoc introduces familiar software environment with pre-optimized OpenCV libraries

Available NOW

Visit the reVISION developer zone
https://www.xilinx.com/products/design-tools/embedded-vision-zone.html#computer

Featured Videos

Introducing reVISION 3:01
reVISION Stack Demo 2:06
4K60 Dense Optical Flow Demo 2:08

View More Videos
## Computer Vision Design Examples

<table>
<thead>
<tr>
<th>Design Example Provided by Xilinx</th>
<th>Latest SDSoC Version Supported</th>
<th>Board &amp; SOM Supported</th>
<th>Provider</th>
</tr>
</thead>
<tbody>
<tr>
<td>LK Dense Optical Flow iterative and pyramidal based implementation doing motion segmentation</td>
<td>2017.1</td>
<td>ZCU102, ZC702, ZC706</td>
<td>Xilinx</td>
</tr>
<tr>
<td>Stereo Disparity Map\nCalculates disparity map from two sensor inputs using local block matching</td>
<td>2017.1</td>
<td>ZCU102, ZC702, ZC706</td>
<td>Xilinx</td>
</tr>
<tr>
<td>Warp Transform</td>
<td>2017.1</td>
<td>ZCU102, ZC702, ZC706</td>
<td>Xilinx</td>
</tr>
<tr>
<td>Harris Corner</td>
<td>2017.1</td>
<td>ZCU102, ZC702, ZC706</td>
<td>Xilinx</td>
</tr>
<tr>
<td>Bilateral Filter</td>
<td>2017.1</td>
<td>ZCU102, ZC702, ZC706</td>
<td>Xilinx</td>
</tr>
</tbody>
</table>
Resources

» INT8 Whitepaper
» Machine Learning Whitepaper
» reVISION Backgrunder
» Additional Papers & Tutorials
» Xilinx Embedded Vision Videos
» Forums

For all this and more, visit Xilinx.com/reVISION
The Embedded Vision Alliance (www.Embedded-Vision.com) is a partnership of 60+ leading embedded vision technology and services suppliers.

Mission: Inspire and empower product creators to incorporate visual intelligence into their products.

The Alliance provides low-cost, high-quality technical educational resources for product developers.

Register for updates at www.Embedded-Vision.com

The Alliance enables vision technology providers to grow their businesses through leads, ecosystem partnerships, and insights.

For membership, email us: membership@Embedded-Vision.com
Learn How to Develop Deep Learning Applications for Computer Vision in TensorFlow

Full-Day, Hands-On Training Class

Topics:
• Introduction to TensorFlow
• TensorBoard Visualization Tools
• Open Source CNN Models
• Neural Networks in TensorFlow
• Object Recognition in TensorFlow
• Using TensorFlow in Embedded Systems

July 13, 2017 • Hyatt Regency Santa Clara • Santa Clara, California
September 7, 2017 • Steigenberger Hotel • Hamburg, Germany

http://bit.ly/2pDRjk4
Q&A