Rosetta: A Realistic High-Level Synthesis Benchmark Suite for Software Programmable FPGAs


Computer Systems Lab
Electrical and Computer Engineering
Cornell University
Increasing Deployment of HLS for FPGAs

Google scholar trend on “HLS for FPGAs”

High demand for deep learning acceleration on FPGAs
Software-programmable FPGA SoCs becoming available

3700+ papers since 2014!
Need for Realistic HLS Benchmarks

- Need for testing a rich set of synthesis directives in modern HLS tools
  - Loop unrolling, pipelining, array partitioning, …

- Need for evaluating new classes of HLS optimization techniques
  - Fast design space exploration [Shao et.al, ISCA’14] [Zhao et.al, ICCAD’17] …
  - Automatic memory banking & reuse [Wang et.al, FPGA’14] [Zhou et.al, FPGA’17] …
  - Polyhedral loop transformations [Pouchet et.al, FPGA’13] [Zhang et.al, FPGA’15] …
Related Work

Popular benchmark suites used by the HLS community

<table>
<thead>
<tr>
<th>Benchmark Suite</th>
<th>Language</th>
<th>Primary Focus</th>
<th>#Benchmarks</th>
<th>Avg. Kernel Length (LOC)</th>
<th>User Directives Applied</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHStone</td>
<td>C</td>
<td>Synthesizability check</td>
<td>12</td>
<td>707</td>
<td>/</td>
</tr>
<tr>
<td>Rodinia</td>
<td>OpenCL</td>
<td>GPU benchmarking</td>
<td>21</td>
<td>356</td>
<td>/</td>
</tr>
<tr>
<td>PolyBench</td>
<td>C</td>
<td>Polyhedral analysis</td>
<td>30</td>
<td>29</td>
<td>/</td>
</tr>
<tr>
<td>MachSuite</td>
<td>C/C++</td>
<td>Kernel selection &amp; optimization</td>
<td>19</td>
<td>88</td>
<td>Unrolling, pipelining, array partitioning, functional unit selection</td>
</tr>
<tr>
<td>Spector</td>
<td>OpenCL</td>
<td>Design space exploration</td>
<td>9</td>
<td>204</td>
<td>Unrolling, pipelining, OpenCL vector type &amp; workgroup size</td>
</tr>
</tbody>
</table>

Existing HLS benchmarks typically lack
1. Large complex applications with realistic design constraints
2. Optimized HLS implementation on real FPGA devices
Rosetta: A Realistic HLS Benchmark Suite

Realistic applications with optimizations and constraints
- Complementary to existing benchmark suites

Retargetable to different FPGA platforms and HLS tools
- Specified in multiple programming languages

Reducible to sub-kernels for micro-benchmarking

Rosetta gets the name following the convention of a plethora of “stone” benchmark suites. It also symbolizes that our benchmarks are specified in multiple languages (i.e., C++, OpenCL) and useful for evaluating HLS across different tools and platforms.

Figure source: https://discoveringegypt.com/egyptian-video-documentaries/mystery-of-the-rosetta-stone/
Open-Source Release on GitHub

- [github.com/cornell-zhang/rosetta](https://github.com/cornell-zhang/rosetta)
Rosetta Overview

3D Rendering

Optical Flow

Face Detection

Digit Recognition

Spam Filtering

Binarized Neural Network
Rosetta Overview

- Programming languages
  ![C++]

- Current targets
  - Cloud FPGA (AWS F1)
  - Embedded FPGA (Xilinx ZC706)

- Synthesizable baseline designs also provided
Software Baseline vs. Optimized Version

- Execution times on ZC706

Rosetta is also useful as a design tutorial on usage of HLS optimization directives
Rosetta Overview

3D Rendering

Optical Flow

Face Detection

Digit Recognition

Spam Filtering

Binarized Neural Network
3D Rendering

- Render images from 3D triangle meshes
- Performance constraint: real-time
- Dataflow optimization

Dolphin figure source: https://opentechschool-brussels.github.io/intro-to-webGL-and-shaders/log1_graphic-pipeline
Challenge: Stage Balancing with Data-Dependent Loops

Balanced dataflow pipeline

// color the frame buffer
void coloringFB(bit16 n_pixels, Pixel* pixels, bit8* buffer) {
  for (bit16 i = 0; i < n_pixels; i ++)
    #pragma pipeline
    buffer[pixels[i].x * MAX_Y + pixels[i].y] = pixels[i].color;
}

Imbalanced dataflow pipeline

Latency is data-dependent
Difficult to balance the pipeline stages without profiling
Digit Recognition

- K-nearest-neighbor digit recognition
  - Downsampling, binarized MNIST subset
    - 14x14 resolution => 196 bits per digit

- Compute kernels
  - Hamming distance calculation: bit-level operation intensive
  - Sorting-based KNN voting

Challenge: Area/Delay Estimation for Bitwise Operations

- Popcount of a wide integer: a subroutine in hamming distance kernel

```c
bit8 popcount(bit196 digit) {
    #pragma pipeline
    bit8 ones = 0;
    for (int i = 0; i < 196; i++)
        ones += digit[i];
    return ones;
}
```

(Narrow) Adder Tree

196 input bits

Observation: Due to lack of awareness of post-RTL optimization, HLS may overestimate delay & resource of bit-level ops, resulting in suboptimal QoR

<table>
<thead>
<tr>
<th>Resource Estimate</th>
<th>LUT</th>
<th>FF</th>
</tr>
</thead>
<tbody>
<tr>
<td>HLS</td>
<td>1505</td>
<td>69</td>
</tr>
<tr>
<td>Post PAR</td>
<td>245</td>
<td>68</td>
</tr>
</tbody>
</table>
Spam Filtering

- Train a logistic regression model for email classification
  - Stochastic gradient descent (SGD)

- Representative kernels in machine learning
  - Dot product, vector addition, sigmoid function
Challenge: Saturating Memory Bandwidth

Default: one feature per cycle
DRAM bandwidth wasted

Optimized: multiple features per cycle
DRAM bandwidth fully utilized

Parameterized: effective DRAM bandwidth

All training samples
Local copy: one sample
Accelerator
A Memory-bound Application

- DRAM: All training samples
  - Features
- FPGA: Local copy: one sample
  - Accelerator

Spam filtering results on AWS

- Kernel Time (ms)
- Vector Kernel Parallelization Factor
- BW = 512 bits
- BW = 256 bits
- BW = 128 bits
- BW = 64 bits
Challenge: Tuning Fixed-point Bitwidth

- Datatype customization with fixed-point types
  - Training samples
  - Parameter vector
  - Sigmoid values
  - Other intermediate results

Require careful tuning of the bitwidths for a good trade-off between model accuracy, throughput and area
Optical Flow

- Computing/estimating motion field from time-varying image intensity

- Compute kernels: Eight-stage of image filtering
  - 1D and 2D stencil access patterns

- Performance constraint: 30 frames/s
Challenge: Data Reuse Across Frames

- Data reuse in the time dimension
  - Optical flow kernel reads from previous four frames

- Buffer previous image frames in DRAM
Challenge: Data Reuse Across Frames

- Data reuse in the time dimension
  - Optical flow kernel reads from previous four frames
- Buffer previous image frames in DRAM
Challenge: Data Reuse Across Frames

- Data reuse in the time dimension
  - Optical flow kernel reads from previous four frames

- Buffer previous image frames in DRAM

Automatic reuse technique required
Face Detection [Srivastava et.al, FPGA’17]

- Viola-Jones face detection algorithm

- Compute kernels
  - Image scaling (memory intensive)
  - Cascaded classifiers (integer arithmetic intensive)

- Performance constraint: 30 frames/s

Figure adopted from: N. Srivastava, FPGA’17 presentation
Challenge: Memory Banking with Irregular Access Patterns

- Irregular memory access pattern of classifiers

```c
int cascadeClassifier( Pixel IntImg[], …) {
    for ( i = 0; i < 25; i++) {
        ...
        for ( j = 0; j < stages_array[i] ; j++) {
            #pragma pipeline
            ....
            c[0] = IntImg[r0.y][r0.x];
            c[1] = IntImg[r0.y][r0.x + r0.w];
            ....
            c[10] = IntImg[pt.y + r2.y + r2.h][pt.x + r2.x];
            c[11] = IntImg[pt.y + r2.y + r2.h][pt.x + r2.x + r2.w];
            ....
        }
    }
}
```

- Apply a customized partitioning scheme
  - Led to a trace-based memory banking work [Zhou et.al, FPGA’17]

Default: Twelve 625x1 muxes
170K LUTs, fail timing

Optimized: Two-level mux network
16K LUTs, timing met

Figure adopted from: Y. Zhou, FPGA’17 presentation
Based on CIFAR-10 BNN model [Courbariaux et.al, arXiv’16]

- Binarized convolution/dense layers
  - Bitwise operation intensive
  - Convolution layers are compute-bound
  - Dense layers are memory-bound

Figure source: R. Zhao, FPGA’17 presentation
Challenge: Shared-Layer Compute Engine

Many diverse sources of parallelism

Effective parallelization of the accelerator

Different network layer types and sizes

Flexibility in accelerator architecture

Large data size and slow communication

Data reuse and storage sharing

Difficult for manual HLS design!
## Experimental Results

### Results on ZC706

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>#LUTs</th>
<th>#FFs</th>
<th>#BRAMs</th>
<th>#DSPs</th>
<th>Runtime (ms)</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D Rendering</td>
<td>8893</td>
<td>12471</td>
<td>48</td>
<td>11</td>
<td>4.7</td>
<td>213 frames/s</td>
</tr>
<tr>
<td>Digit Recognition</td>
<td>41238</td>
<td>26468</td>
<td>338</td>
<td>1</td>
<td>10.6</td>
<td>189k digits/s</td>
</tr>
<tr>
<td>Spam Filtering</td>
<td>12678</td>
<td>22134</td>
<td>69</td>
<td>224</td>
<td>60.8</td>
<td>370k samples/s</td>
</tr>
<tr>
<td>Optical Flow</td>
<td>42878</td>
<td>61078</td>
<td>54</td>
<td>454</td>
<td>24.3</td>
<td>41.2 frames/s</td>
</tr>
<tr>
<td>Face Detection</td>
<td>62688</td>
<td>83804</td>
<td>121</td>
<td>79</td>
<td>33.0</td>
<td>30.3 frames/s</td>
</tr>
<tr>
<td>BNN³</td>
<td>46899</td>
<td>46760</td>
<td>102</td>
<td>4</td>
<td>4995.2</td>
<td>200 images/s</td>
</tr>
</tbody>
</table>

1: K=3, PAR_FACTOR=40. 2: Five epochs, PAR_FACTOR=32, VDWIDTH=64. 3: Eight convolvers, 1000 test images.

### Results on AWS F1

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>#LUTs</th>
<th>#FFs</th>
<th>#BRAMs</th>
<th>#DSPs</th>
<th>Runtime (ms)</th>
<th>Throughput</th>
<th>Performance-cost Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D Rendering</td>
<td>6763</td>
<td>7916</td>
<td>36</td>
<td>11</td>
<td>4.4</td>
<td>227 frames/s</td>
<td>496k frames/$</td>
</tr>
<tr>
<td>Digit Recognition</td>
<td>39971</td>
<td>33853</td>
<td>207</td>
<td>0</td>
<td>11.1</td>
<td>180k digits/s</td>
<td>393M digits/$</td>
</tr>
<tr>
<td>Spam Filtering</td>
<td>7207</td>
<td>17434</td>
<td>90</td>
<td>224</td>
<td>25.1</td>
<td>728k samples/s</td>
<td>1.6G samples/$</td>
</tr>
<tr>
<td>Optical Flow</td>
<td>38094</td>
<td>63438</td>
<td>55</td>
<td>484</td>
<td>8.4</td>
<td>119 frames/s</td>
<td>260k frames/$</td>
</tr>
<tr>
<td>Face Detection</td>
<td>48217</td>
<td>54206</td>
<td>92</td>
<td>72</td>
<td>21.5</td>
<td>46.5 frames/s</td>
<td>101k frames/$</td>
</tr>
</tbody>
</table>

1: K=3, PAR_FACTOR=40. 2: Five epochs, PAR_FACTOR=32, VDWIDTH=512.
Conclusions and Future Work

- Rosetta can serve as
  - A realistic benchmark suite for the HLS research community
    • Complex applications with rich compute/memory characteristics
    • Synthesizable baseline designs exposing opportunities to new HLS automation
  - A useful design tutorial for HLS/FPGA users
    • Optimized implementations meeting realistic design constraints

- Future work
  - Incorporate more applications from different fields
  - Provide OpenCL version for each benchmark
  - Continue to optimize the applications
Rosetta: A Realistic High-Level Synthesis Benchmark Suite for Software Programmable FPGAs


github.com/cornell-zhang/rossetta