Motivation

Have Python, need acceleration. As modern FPGAs evolve with more heterogeneous processing components they really are processors first and FPGA accelerators second. Python is a rapidly growing language and is used for everything from simple scripting to complex parallel computation. It has grown in popularity from #4 in 2014 to #1 in the 2017 IEEE Spectrum Ranking[4]


<table>
<thead>
<tr>
<th>Language Name</th>
<th>Types</th>
<th>Spectrum Ranking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python</td>
<td>2.7</td>
<td>1</td>
</tr>
<tr>
<td>Java</td>
<td>2.4</td>
<td>2</td>
</tr>
<tr>
<td>C++</td>
<td>2.7</td>
<td>3</td>
</tr>
<tr>
<td>4.4</td>
<td>2.7</td>
<td>4</td>
</tr>
<tr>
<td>2.0</td>
<td>2.7</td>
<td>5</td>
</tr>
</tbody>
</table>

sPyC Python-to-C

A source-to-source translation tool for Python syntax to HLS-C
- Goal: Use pure Python syntax
  - Using unmodified Python AST parser, no custom syntax
  - Any sPyC-specific syntax must not modify original behavior
- Additional requirements beyond basic Python
  - Function arguments, return type, must be annotated[2]
  - Variables may be annotated[1]

Supported Syntax
- Primitive variables (char,short,int,long,float,double)
- Multidimensional Numpy arrays
- Loops (for/while), conditionals (if/elif/else)
- Arithmetic operations (+,-,*,/)
- Sub-functions
- Built-in – range(), ndarray.shape, Numpy functions
- Pragmas via Python docstrings

Supported Transformations
- Multidimensional to single dimension array conversion
- Limited variable type inference

Python Accelerator Design Flow
1. Translate Python function to C
2. EDA Implementation flow
  - Implement accelerator with HLS
  - Generate bitstream
  - Generate accelerator drivers
3. Generate Python C API wrapper
4. Rewrite App to use accelerator
  - Contiguous ndarray wrap
  - Call new accelerator driver

Performance Results
- Canny edge detection on 320x240 images
- Running on Pynq board (Zynq-7020) @ 650MHz
- Using 100MHz accelerator clock

<table>
<thead>
<tr>
<th>App</th>
<th>Performance</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original Python</td>
<td>48.14 sec</td>
<td>1.0x</td>
</tr>
<tr>
<td>Refactored Python</td>
<td>139.28 sec</td>
<td>0.3x</td>
</tr>
<tr>
<td>Unoptimized HLS</td>
<td>58.68 ms</td>
<td>820.0x</td>
</tr>
<tr>
<td>Pipelined HLS</td>
<td>12.22 ms</td>
<td>3,939.0x</td>
</tr>
<tr>
<td>Partitioned HLS</td>
<td>1.23 ms</td>
<td>39,137.0x</td>
</tr>
<tr>
<td>OpenCV (cv2)</td>
<td>7.19 ms</td>
<td>6,695.0x</td>
</tr>
</tbody>
</table>

PYNQ[1] – A Python development environment for application acceleration on Xilinx Zynq devices. Provides: hardware, libraries, APIs, and design methodologies.

Example Python App

```python
while True:
    frame = hdmi_in.readframe()
    if cnt < 10 * 5:
        out = hdmi_out.newframe()
        invert(frame, out)
    hdmi_out.writeframe(out)
else:
    hdmi_out.writeframe(frame)
```

Bridging the Gap

In order to use Python as the main development language with existing tools and frameworks, additional support is needed to provide an end-to-end flow.
- **Pylon** – generates Python C API wrapper/binding to call accelerators
- **Pyramid** – generates script to manage EDA implementation flow
- **Pyrite** – rewrites original source App to import accelerator module and call new driver function

[1] PEP 526 – Syntax for Variable Annotations
[2] PEP 484 – Type Hints