NANYANG TECHNOLOGICAL UNIVERSITY
School of Computer Engineering

Accelerator Design at High Level of Programming Abstraction

Prashant Ravi, Abhishek Kumar Jain, Suhaib A. Fahmy, Douglas L. Maskell
prashant014@ntu.edu.sg, abhishek013@ntu.edu.sg, sfahmy@ntu.edu.sg, asdouglas@ntu.edu.sg

Background and Motivation

Accelerator Design Productivity Issues:

- Difficulty of accelerator design at low level
- Complex Host-Accelerator interfacing
- Lack of software like abstractions
- Long compilation times (Place and route)

One solution: High Level Synthesis (HLS)

- Accelerator design in a high level language
- Automated host-accelerator interfacing
- RTL generation from C/C++/OpenCL
- Benefits of software like abstractions
- SDSoC, **AOCL**, SDAccel
- Long compilation time still an issue

Coarse-Grained FPGA Overlays:

- Fast compilation and development cycles
- Easy to use even by novice programmers
- Access of FPGA for software developers
- **MXP**, GRVI-Phalanx, DeCO, DySER

Contributions

- Analysis of Overlay architectures like Vectorblox MXP and Altera OpenCL SDK as alternatives to pure RTL design flow. Highlighting the ease of use and fast learning curves of these methods.
- Benchmarking and comparison of timing performance as well as operations per cycle for the Vectorblox MXP processor and the Altera OpenCL Implementation with arm CPU implementation.

Observations

The Vectorblox MXP soft vector processor:

- Extremely short learning curve
- The Vectorblox MXP C/C++ api is extremely easy to use.
- Full control of DMA and execution to programmer.
- Compilation and debugging time equal to that of traditional C/C++ debugging.

The Altera OpenCL SDK:

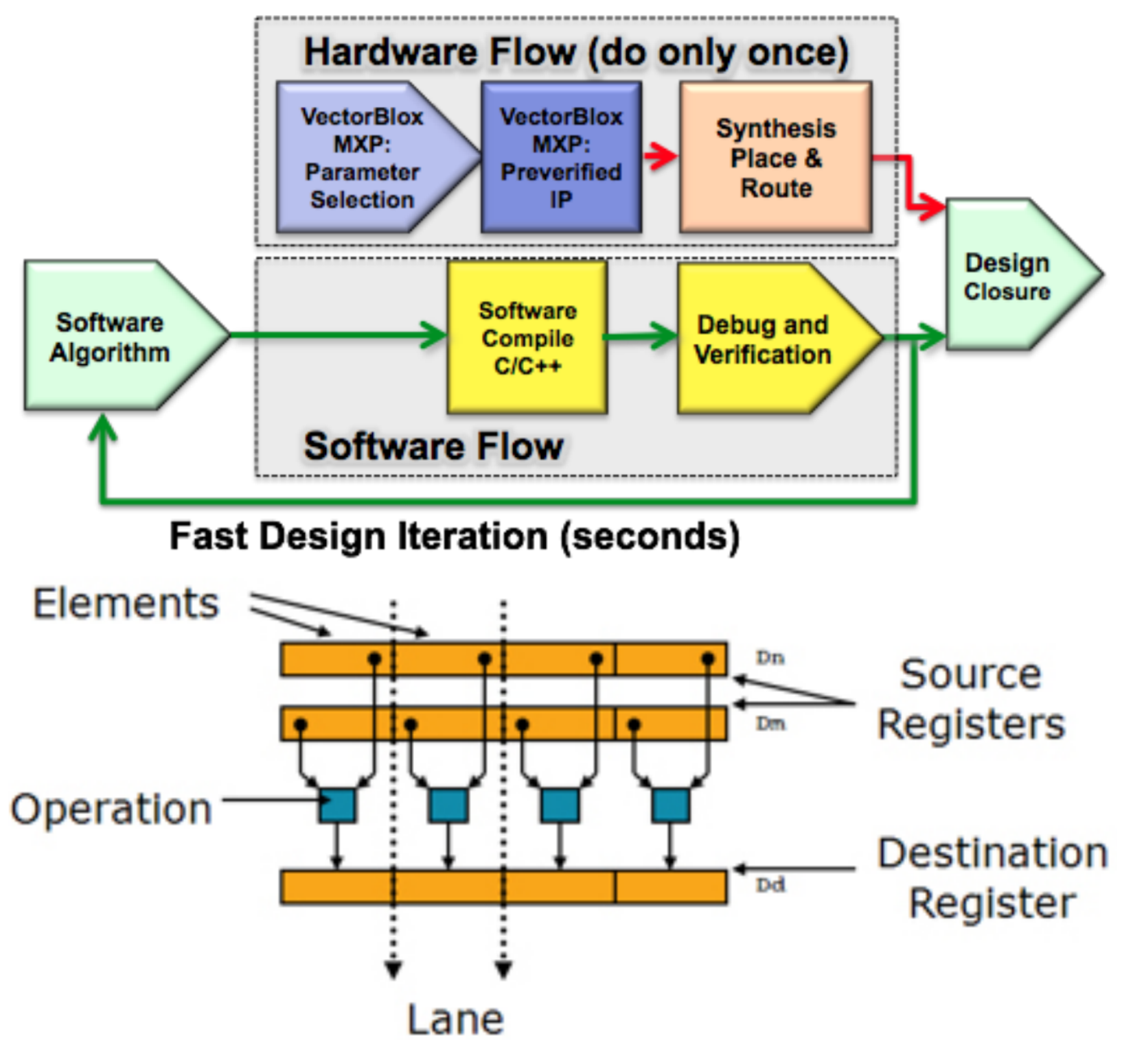
- Minimal modification of traditional OpenCL code required.
- kernel optimizations like vectorization, hardware replication for parallelism.
- Dynamic re-programming of FPGA at runtime with totally new kernel.
- Host code portability across all devices.

Conclusions and Future Work

- More popularity of overlays and high level synthesis tools
- Place and route effeciency improvements in OpenCL to hardware
- Overall generated hardware effeciency improvment
- More awareness in the software world and amongst software developers

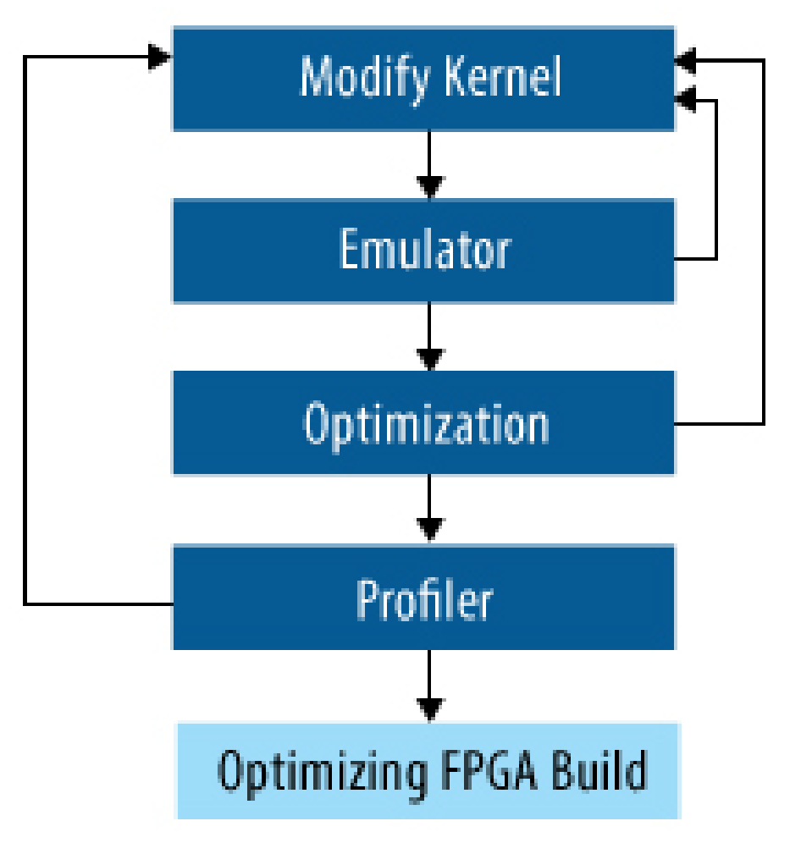
The Vectorblox MXP Processor

- Accelerating Bare metal C/C++ applications on Zynq using MXP
- Easy to use software APIs for embedded vector processing
- 16 vector lanes each containing an ALU
- Good for integer operations, lacks floating point



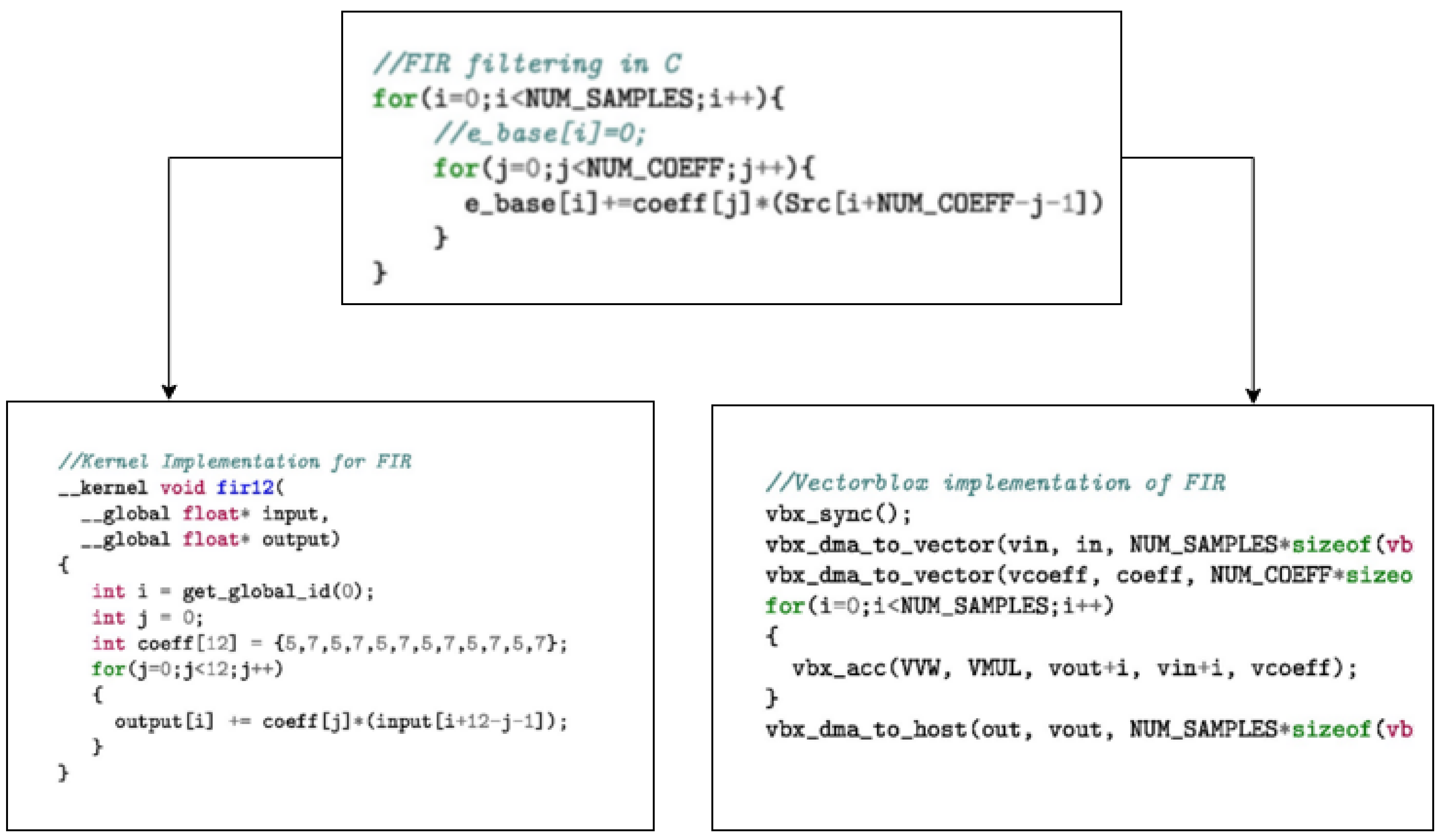
DE0-NANO-SoC with OpenCL

- Creation of a semi-customized pipelined datapath using OpenCL Kernels.
- Full control of hardware replication at the kernel level with loop unrolling.
- Functional C emulator for functional verification wihout generating hardware.
- Special AOCL optimizations for floating point to reduce area.
- Dedicated AOCL memory channels for data transfer to and from accelerator.



Quick Easy Mapping of Compute Kernels

- Utilization of C Api very straightforward in case of Vectorblox.
- Translation of C code to OpenCL and VBX code fairly straightforward
- Placement and Routing required in case of AOCL, and only compilation in case of vectorblox.



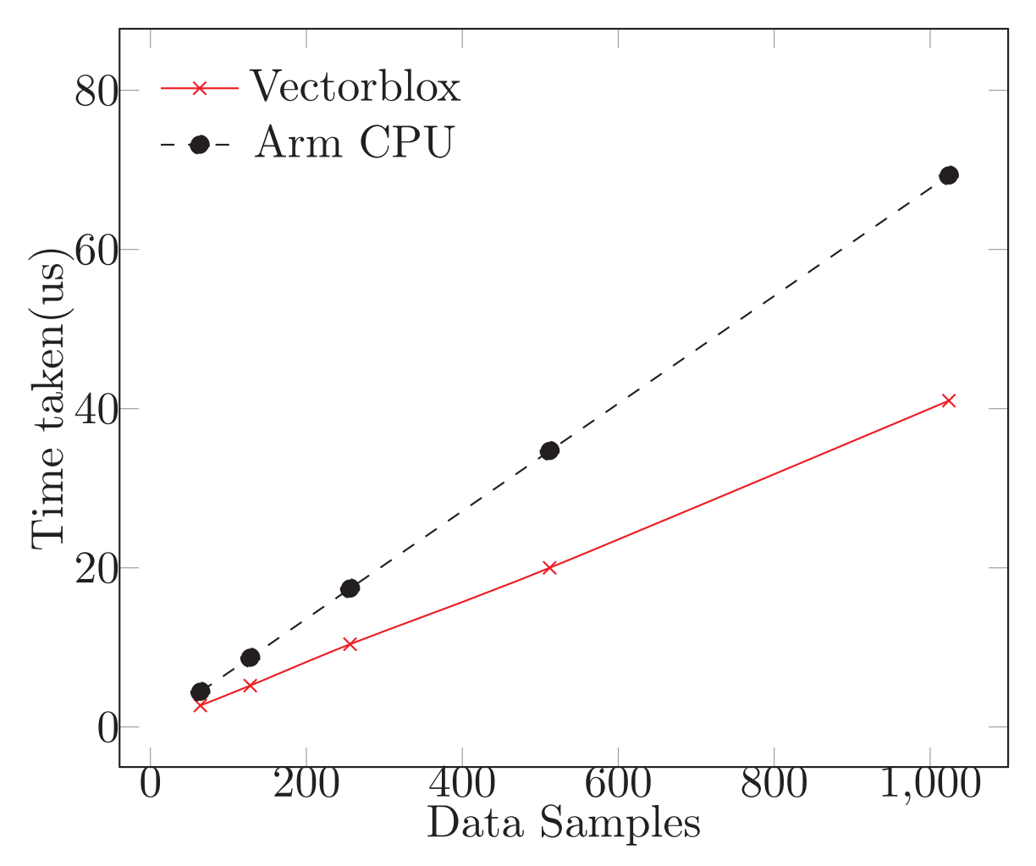
```
//FIR filtering in C
for(i=0;i<NUM_SAMPLES;i++){
    //e_base[i]=0;
    for(j=0;j<NUM_COEFF;j++){
      e_base[i]+=coeff[j]*(Src[i+NUM_COEFF-j-1])
    }
}
```

```
//Kernel Implementation for FIR
__kernel void fir12(
  __global float* input,
  __global float* output)
{
   int i = get_global_id(0);
   int j = 0;
   int coeff[12] = {5,7,5,7,5,7,5,7,5,7,5,7};
   for(j=0;j<12;j++)
   {
     output[i] += coeff[j]*(input[i+12-j-1]);
   }
}
```

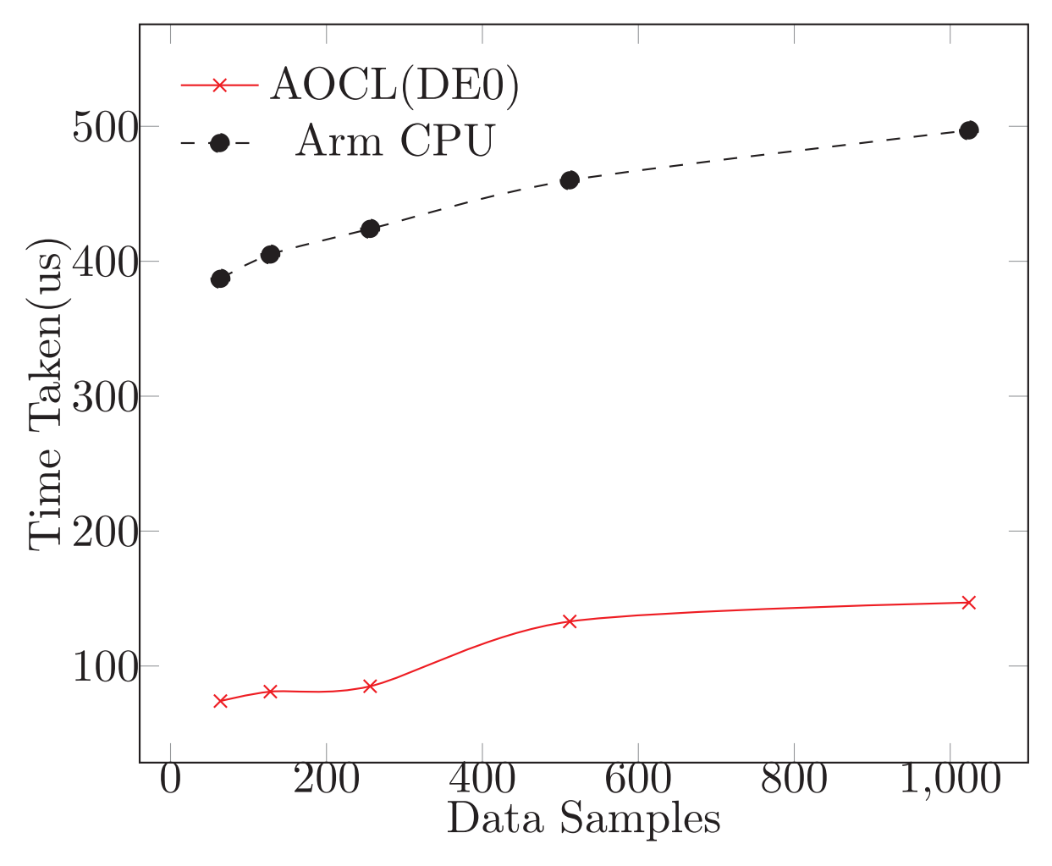
```
//Vectorblox implementation of FIR
vbx_sync();
vbx_dma_to_vector(vin, in, NUM_SAMPLES*sizeof(vb
vbx_dma_to_vector(vcoeff, coeff, NUM_COEFF*sizeo
for(i=0;i<NUM_SAMPLES;i++)
{
  vbx_acc(VVW, VMUL, vout+i, vin+i, vcoeff);
}
vbx_dma_to_host(out, vout, NUM_SAMPLES*sizeof(vb
```

Experimental Evaluation

- 12-Tap FIR filter kernel was executed on vectorblox, ARM CPU on Zedboard and on the Cyclone V FPGA with AOCL and the timing performance was compared.
- The Vectorblox and AOCL implementations ran faster than the ARM but when comapred to AOCL, Vectorblox performed much faster.



Vectorblox C API Implementation in Comparison with naive C Implementation on ARM CPU.



OpenCL Implementation Timing Comparison between ARM CPU and AOCL FPGA Implementation.

- The MXP gave a **1.6x Operations Per Cycle** increase over ARM and a **1.5x Speedup**.
- The AOCL implementation gave **3.3x Operations per Cycle** increase over ARM with a **4x Speedup**.