Hardware Acceleration Without The Use Of RTL

NANYANG TECHNOLOGICAL UNIVERSITY
School of Computer Engineering

Prashant Ravi
prashant014@ntu.edu.sg

Background and Motivation

Major issues in mainstream adoption of FPGAs:

- Difficulty of accelerator design at low level
- Long compilation times (Place and route)
- Poor design productivity

One possible solution is to use FPGA Overlay:

- Accelerator design in a high level language
- Fast compilation and development cycles
- Easy to use even by novice programmers
- Puts the FPGA in the hands of software developers.

Another solution is to use novel high level synthesis tools:

- HLS tools to generate C to RTL, eg:Vivado HLS, SDSoC
- OpenCL to hardware design synthesis eg:AOCL, SDAccel

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:

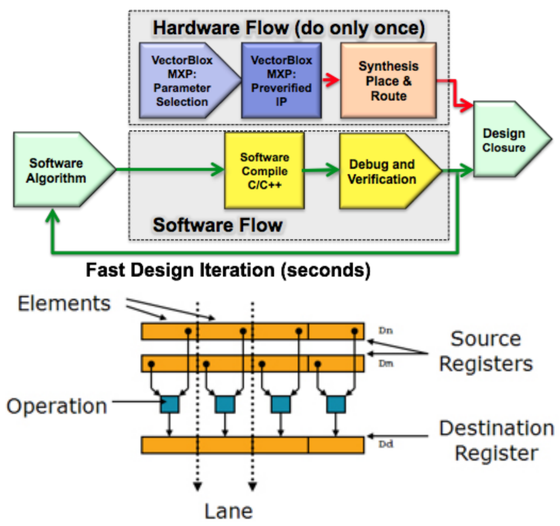
- Minimal modification of traditional OpenCL code required.
- kernel optimizations like vectorization, hardware replication for parallelism.
- Dynamic re-programming of FPGA at runtime with totally new kernel.
- Host code portability across all devices.

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.

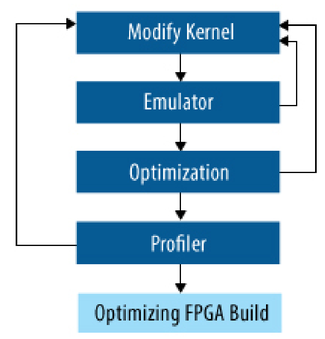
The Vectorblox MXP Processor

- Has a number of vector lanes containing an ALU in each lane.
- Parallel processing on multiple vector input elements.
- Extremely good for integer operations, lacks floating point.
- Customizable vector lanes and option to add custom instructions.



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])
    }
}
```

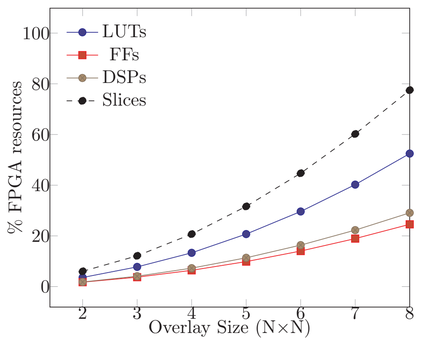
```
//Kernel Implementation for FIR
__kernel void fir12(
  __global float* input,
  __global float* output)
{
   int i = get_global_id(0);
   int j = 0;
   int coeff[12] = {5,7,5,7,5,7,5,7,5,7,5,7};
   for(j=0;j<12;j++)
   {
     output[i] += coeff[j]*(input[i+12-j-1]);
   }
}
```

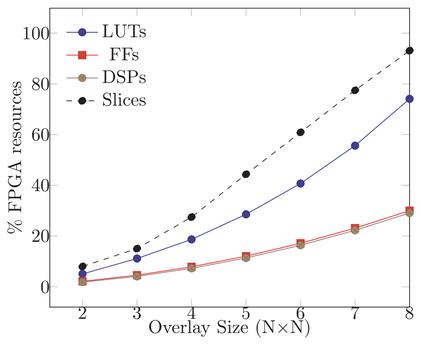
```
//FIR filtering in C
for(i=0;i<NUM_SAMPLES;i++){
    //e_base[i]=0;
    for(j=0;j<NUM_COEFF;j++){
      e_base[i]+=coeff[j]*(Src[i+NUM_COEFF-j-1])
    }
}
```

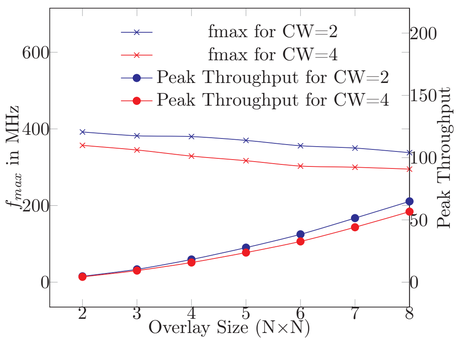
```
//Vectorblox implementation of FIR
vbx_sync();
vbx_dma_to_vector(vin, in, NUM_SAMPLES*sizeof(vb
vbx_dma_to_vector(vcoeff, coeff, NUM_COEFF*sizeo
for(i=0;i<NUM_SAMPLES;i++)
{
  vbx_acc(VVW, VMUL, vout+i, vin+i, vcoeff);
}
vbx_dma_to_host(out, vout, NUM_SAMPLES*sizeof(vb
```

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.







Benchmark	Benchmark Characteristics i/o nodes	op nodes	merged nodes	savings	Routability CW=2	CW=4	Overlay Results Latency	MLI	**GOPS**	HLS Implementation Results Latency	Fmax	**GOPS**	Slices	DSPs
chebyshev	1/1	7	5	28%	3×3	3×3	49	36	**2.59**	13	333	**2.3**	24	3
sgfilter	2/1	18	10	44%	4×4	4×4	54	31	**6.66**	11	278	**5.0**	40	8
mibench	3/1	13	6	53%	3×3	3×3	47	35	**4.81**	9	295	**3.8**	81	3
qspline	7/1	26	22	15%	5×5	5×5	76	64	**9.62**	21	244	**6.3**	126	14
poly1	2/1	9	6	33%	3×3	3×3	34	22	**3.33**	12	285	**2.56**	62	4
poly2	2/1	9	6	33%	3×3	3×3	29	7	**3.33**	11	295	**2.65**	45	4
poly3	6/1	11	7	36%	3×3	3×3	31	11	**4.07**	12	250	**2.75**	52	6
poly4	5/1	6	3	50%	2×2	2×2	24	12	**2.22**	7	312	**1.87**	36	3
atax	12/3	60	36	40%	—	6×6	72	58	**18.0**	13	263	**15.8**	78	18
bicg	15/6	30	18	40%	—	6×6	46	32	**9.0**	7	270	**8.1**	91	18
trmm	18/9	54	36	33%	—	7×7	58	30	**16.2**	8	222	**11.9**	105	36
syrk	18/9	72	45	37%	—	7×7	41	19	**21.6**	10	250	**18**	237	24