A convolutional layer acccelerator implemented using Vivado HLS

Lin Qiao (James) Liu, 1003942807
Department of Electrical and Computer Engineering
University of Toronto
Email: linqiao.liu@mail.utoronto.ca

***Abstract*—The abstract goes here. FILL ME IN!!!!**

I. Introduction

[Give an overall architecture overview in this section, and list the optimizaitons implemented] [1].

II. Implementation Details

[More details on the flow of data and the use of each component in the architecture]

III. Results

[Resource utilization and run-time for 1 batch (batch 0, 10 image)]

sfsfsdf

IV. Conclusion

The conclusion goes here.

Acknowledgment

The authors would like to thank...

References

[1] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *CoRR*, vol. abs/1409.1556, 2014. [Online]. Available: http://arxiv.org/abs/1409.1556

for $(iterK = 0;\ iterK < K;\ iterK+=K_{vec}) \longrightarrow$ **Loop K**
{
  Load K_{vec} biases for $M \times N \times K_{vec}$ partial sums from memory to on-chip buffer;
  for $(iterC = 0;\ iterC < C;\ iterC+=C_{vec}) \longrightarrow$ **Loop C**
  {
    Load $H \times W \times C_{vec}$ inputs from memory to on-chip buffer, perform zero-padding if necessary;
    Load $K_{vec} \times (R \times S \times C_{vec})$ kernel weights from memory to on-chip compute registers;
    for $(iterM = 0;\ iterM < M;\ iterM++) \longrightarrow$ **Loop M**
    {
      for $(iterN = 0;\ iterN < N;\ iterN++) \longrightarrow$ **Loop N**
      {
        Load $H_{vec} \times W_{vec} \times C_{vec}$ inputs from on-chip buffer to compute registers;
        Update K_{vec} partial sums in parallel. Each update dot-multiplies $H_{vec} \times W_{vec} \times C_{vec}$ inputs with weights;
      }
    }
  }
}

Fig. 1. Computation flow for processing one image in the convolutional layer accelerator.

TABLE I
COMPARISION OF DIFFERENT SYSTEMS' RUN TIME FOR FORWARDING 1 BATCH OF 10 IMAGES THROUGH EACH CONVOLUTIONAL LAYER OF THE VGG-16 NETWORK.

Layer	CPU		Array partition and parallel			Parallel without partition			Ping-pong			Array partition and parallel, Dummy
	Time per batch (s)	RMS	Time per batch (s)	RMS	Speedup	Time per batch (s)	RMS	Speedup	Time per batch (s)	RMS	Speedup	Time per batch (s)
conv1_1	6.4	1.8e−11	6.2	4.8e−11	1.03	12.2	4.8e−11	1.03	6.34	4.8e−11	1.01	5.21
conv1_2	114.2	2.7e−9	35.3	1.8e−9	3.23	144.4	1.8e−9	0.79	57.3	1.8e−9	1.99	8.34
conv2_1	57.6	8.6e−9	18.19	5.2e−9	3.17	73.2	5.2e−9	0.79	29.20	5.2e−9	1.97	4.18
conv2_2	114.2	3.5e−8	34.20	2.2e−8	3.34	143.22	2.2e−8	0.80	56.22	2.17e−8	2.03	6.22
conv3_1	56.57	4.7e−8	18.11	3.4e−8	3.12	73.12	3.4e−8	0.77	29.12	3.4e−8	1.94	3.12
conv3_2	112.1	7.4e−8	35.14	5.6e−8	3.19	144.15	5.6e−8	0.77	57.12	5.6e−8	1.96	4.14
conv3_3	112.9	6.1e−8	35.1	4.7e−8	3.21	144.2	4.7e−8	0.78	57.14	4.72e−8	1.98	4.14
conv4_1	56.4	3.6e−8	19.12	2.89e−8	2.95	74.17	2.9e−8	0.76	32.1	2.9e−8	1.76	3.12
conv4_2	113.1	2.0e−8	37.2	1.2e−8	3.04	146.2	1.3e−8	0.77	62.15	1.3e−8	1.82	4.14
conv4_3	113.4	3.3e−9	37.1	2.1e−9	3.05	146.2	2.1e−9	0.78	62.1	2.1e−9	1.82	4.15
conv5_1	28.8	1.7e−9	12.1	1.2e−9	2.39	39.1	1.2e−9	0.74	18.07	1.2e−9	1.59	2.07
conv5_2	29.8	3.7e−10	12.1	2.6e−10	2.5	39.1	2.6e−10	0.76	18.1	2.6e−10	1.65	2.07
conv5_3	28.8	3.7e−11	12.07	2.6e−11	2.4	39.1	2.6e−11	0.74	18.1	2.6e−11	1.60	2.07

TABLE II
RESOURCE UTILIZATION

Resources	Available	Array partition and parallel		Parallel without array partition		Dataflow	
		Usage	Utilization (%)	Usage	Utilization (%)	Usage	Utilization (%)
CLB	201600						
LUT		20342	10.1	18850	9.4	24800	12.3
Registers		34022	8.4	33265	8.3	36688	9.1
RAM18K	1296	1154	89.0	1154	89.0	1282	98.9
DSP	288	232	80.6	239	83.0	245	85.1

```
void util_computeKernel (
t_conv (&partialOutputBuffer)[K_vec],
t_conv (&computeCache)[K_vec][H_vec × W_vec × C_vec],
t_conv (&computeStream)[H_vec × W_vec × C_vec] )
{
 COMPUTE_FOR_OUTPUT:
 for (unsigned int i = 0; i < K_vec; i + +)
 {
  #pragma HLS UNROLL
  for (unsigned int j = 0; j < H_vec × W_vec × C_vec; j++)
  {
   COMPUTE_FOR_DOT_PRODUCT:
   #pragma HLS UNROLL
   partialOutputBuffer)[i]
   +=computeCache[i][j] × computeStream[j];
  }
 }
}
```

Fig. 2. HLS Code for the computation engine.