Convolutional Nets on FPGA – Checkpoint

Ricson Cheng — `ricsonc`
Sree Vishant Prabhakaran — `sreevisp`

April 8, 2017

Work completed so far

1. We implemented pooling, padding, reshaping, and ReLu modules in Verilog
2. We implemented a convolution module in Verilog, supporting multi-channel inputs and multiple filters.
3. We designed and trained an 11-layer convolutional neural network to classify images from the CIFAR-10 dataset in Keras, since we plan take the weights from this GPU-trained model and put them on the FPGA. The final train accuracy was 82%.
4. We quantized the weights, which were originally float32 type, to int32 type, so that they could be used in Verilog
5. We finished a baseline implementation of our convolutional neural network in Verilog
6. We wrote a test module and can use it to test the network

Deliverables

The list of deliverables from the proposal is replicated below for convenience:

1. Implement a fully convolutional network on FPGA (This is our final goal, after all)
2. Optimize code until it is significantly faster (5x) than our baseline. We expect this to be possible due to similar speedup factors seen in our readings.
3. Achieve good classification accuracy on the MNIST dataset, at least 95%. This should be possible because the MNIST dataset is comparatively easy for convolu-

tional nets to do well on. We may also try more challenging datasets if time allows, however this is not the focus of the project.

4. Adapt the work of [1] in our implementation. We expect this to speed up our implementation because single-bit computations may be much faster than dealing with 32-bit floating point numbers.
5. Analyze the speedup achieved due to different optimizations and data transformations.

We have completed items 1 and 3 (albeit on CIFAR-10 instead of MNIST). We still expect to be able to do items 2 and 5. We do not plan on completing 4 anymore, because we decided it was not very relevant to our project. We also add the following new goals to our project:

1. Reusing initialized modules: Currently, modules are not reused. That is, the convolution for each layer is done using a different part of the circuitry, instead of reusing circuitry.
2. Pipelining: Depending on the implementation design, it may be possible to start processing a second image before the first image is finished. This would increase throughput of the implementation.

1 Will be a "planned" goal and 2 will be a "hope to achieve" goal.

Parallelism competition

We plan to show performance graphs and an explanation of our implementation at the parallelism competition.

Preliminary Results

We classified 9 out of 10 images correctly, chosen randomly from the training set.

Issues

The biggest difficulty we have encountered so far is moving data (images and network weights) onto the FPGA. Currently, we are doing the naive thing, and just creating modules in Verilog which contain just a huge table of values. This is working OK so far, but is not ideal, and the software does not support arrays with size over 1 million bits, which may become problematic if we want to use larger networks or larger images.

Revised Schedule

Week 3.0 (April 24 - 28)

- Figure out how to use block ram
- Place weights and images on the block ram
- Run baseline model on FPGA

Week 3.5 (April 29 - April 30)

- Implement convolutions in the “reusable module” form.
- Read some recent papers in more depth to understand possible challenges

Week 4.0 (May 1 - May 5)

- Implement padding, relu, and pooling in reusable module form
- Train and run a larger network

Week 4.5 (May 6 - May 7)

- Optimize the convolution module, which will be the most compute heavy.
- Possibly add pipelining into the system

Week 5.0 (May 8 - 12)

- Wrap up any remaining bugs/features/optimizations
- Prepare final presentation