Custom Acceleration with FPGAs

CS5222 Advanced Computer Architecture

Simone Mezzaro

Part 1: Matrix Multiplication Pipeline Optimization in HLS

B. Pipelining in HLS

- Design latency: 13810 cycles
- Device utilization:

	BRAM_18K	DSP48E	FF	LUT
Total	16	10	25329	38962
Available	280	220	106400	53200
Utilization (%)	5	4	23	73

- Number of floating point adders: 2
  Number of floating point multipliers: 2
- Initiation interval of pipelined loops:

Loop name	Initiation interval
LOAD_OFF_1	1
LOAD_W_2	1
LOAD_I_2	1
L2	128
STORE_O_1	5

C. Increasing Pipeline Parallelism by Repartitioning Memories

- Design latency: 4962 cycles
- Device utilization:

	BRAM_18K	DSP48E	FF	LUT
Total	36	80	37945	35357
Available	280	220	106400	53200
Utilization (%)	12	36	35	66

- Number of floating point adders: 16
  Number of floating point multipliers: 16
- Initiation interval of pipelined loops:

Loop name	Initiation interval
LOAD_OFF_1	1
LOAD_W_2	1
LOAD_I_2	1
L2	16
STORE_O_1	5

D. Amortizing Iteration Latency with Batching

- Design latency: 78370 cycles
- Device utilization:

	BRAM_18K	DSP48E	FF	LUT
Total	154	80	38171	35778
Available	280	220	106400	53200
Utilization (%)	55	36	35	67

E. Extending Batch Size with Tiling

- Design latency: 627959 cycles
- Device utilization:

	BRAM_18K	DSP48E	FF	LUT
Total	86	80	38234	35844
Available	280	220	106400	53200
Utilization (%)	30	36	35	67

F. Hardware compilation and FPGA testing on the PYNQ

- Measured speedup: 7.78×
- Measured accuracy: 86.96%

Part 2: Fixed-Point Optimizations

- Fixed-point validation accuracy reported by mnist.py: 82.26%
- Design latency: 387707 cycles
- Device utilization:

	BRAM_18K	DSP48E	FF	LUT
Total	4	129	22290	24936
Available	280	220	106400	53200
Utilization (%)	1	58	20	46

- Measured speedup: 57.21×
- Measured accuracy: 81.49%
- Number of multipliers: 127
- Initiation interval of L2 loop: 1 cycle
- The design is bandwidth-limited. The bottleneck is the pipelined loop LOAD_I_1, which has an initiation interval of 32 cycles.

Part 3: Open-ended design optimization

Optimization 1

The first optimization performed is image resizing. Input images have been resized from 16×16 to 12×12, bringing the number of features down to 144. This change allowed to reduce the initiation interval of the bottleneck loop LOAD_I_1 to 18. The new design also required to set the array partitioning factor to 72.

This implementation has a total latency of 273429 cycles and grants enough speedup to achieve less then $4.5ms$ of FPGA inference. Performances measured on the board are:

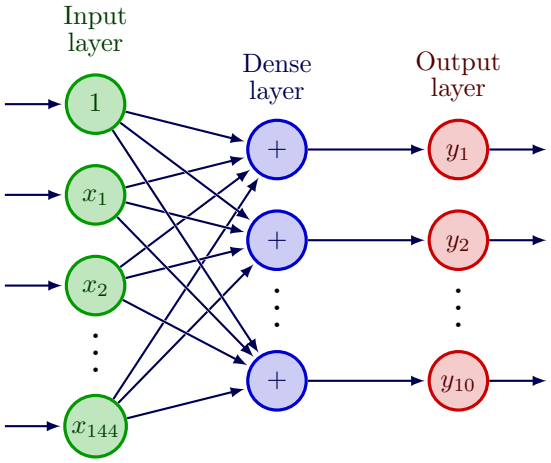
- Time: $4.1ms$
- Accuracy: 80.68%

Notice that it is also possible to reduce the inference time below $3ms$ by resizing the inputs to 8×8, but degrading the accuracy of an additional $\approx 3\%$.

The image resizing to 12×12 is kept also in the following optimizations.

Optimization 2

The second optimization introduces the use of TensorFlow python package to implement and train a neural network for images classification. The designed neural network is based on Perceptrons. Its structure is shown in the following graph.



$x_1, x_2, ..., x_{144}$ are the input values of the image (the features). The dense layer is composed of 10 Perceptrons, one for each class. The output of the j-th Perceptron is

$$out_j = offset_j + \sum_{i=1}^{144} w_{ij} \cdot x_i$$

The outputs of the Perceptons are then normalized to obtain the final outputs $y_1, y_2, ..., y_{10}$.

The main advantage of this simple network is that it does not require any change to the hardware design. The trained weights and offsets can be extracted from the Dense layer, quantized and passed directly to the board as before.

Post training quantization is performed in a similar way to previous implementations, with a few changes.

- The $SCALE$ factor is now computed as

  $$SCALE = \frac{255}{max\_trained - min\_trained}$$

  where $max\_trained$ and $min\_trained$ are respectively the maximum and the minimum value between all the trained weights and offsets.

- A $ZERO\_POINT$ parameter has been introduced to center the interval of trained values around zero.

  $$ZERO\_POINT = \frac{max\_trained + min\_trained}{2}$$

- Each quantized value is now computed as

  $$quant\_value = (flaot\_value - ZERO\_POINT) \cdot SCALE$$

In this way the values after quantization are distributed along the entire available interval $(-128, 127)$.

Performances measured on the board are:

- Average time: $4.1ms$
- Average accuracy: 87.48%
- Best accuracy: 88.54%
- Worst accuracy: 85.84%

Notice that now the accuracy changes every time the classifier is trained because the weights are initialized at random. For this reason both average, best and worst accuracy have been reported.

This second optimization has an average improvement of $\approx$ 7% over the previous one and meets the accuracy requirement.

Optimization 3

The last optimization modifies *optimization 2* by introducing quantization aware training in the neural network. This technique incorporates quantization in the training process: every time the dense layer need to be executed, the floating point weights are quantized to obtain integer weights which are then used by the Perceptrons. Once the output of the network is obtained, the original floating point weights are trained. In this way the model takes into consideration the errors due to quantization already during the training phase.

Quantization aware training has been performed using predefined classes from the TensorFlow package. On the other hand, the post training quantization is still performed as in *optimization 2*.

Performances measured on the board are:

- Average time: $4.1ms$
- Average accuracy: 86.76%
- Best accuracy: 88.27%
- Worst accuracy: 85.86%

This last optimization meets the accuracy requirement, but shows no improvement with respect to the second one. The lack of improvement can attributed to two aspects of the design:

- The simplicity of the neural network represents a bottleneck: having a more complex but more powerful network could have shown an increasing accuracy when combined with quantization aware training.
- The quantization used during the training process is not the exact same used for post training quantization. Therefore the final quantization introduces errors for which the model has not been trained.