DSA

林品妤

***Abstract*—This report discusses about the mechanism of MLP evaluation and how IP communicates with Aquila. And, some improvement is implemented to accelerate the execution.**

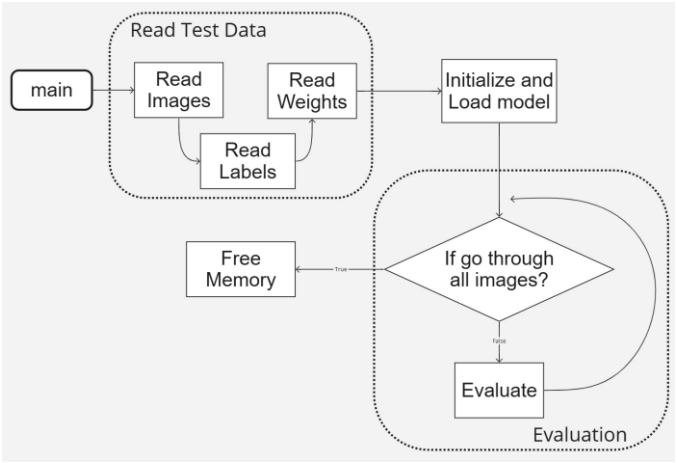
Keywords—Aquiila; MLP; IP integration;

I. INSIGHT TO OCR

The goal of `ocr` program is to perform hand-written digits recognition based on provided images and trained weights. The whole program workflow is shown as Fig 1. At the beginning, it read data including images, labels, and weights from SD card into heaps. After loading essential data, it initializes and loads neuro network model. Then, go through all images to perform evaluation for hand-written digits recognition. At the end, free all memory resources.

Know that the key hotspot of the program appears inside the function `neuronet_eval` called during evaluation. Before accelerate the program, we need to understand the mechanism of the neuro network NeuroNet, including how data are stored and how the evaluation works.

Fig. 1. Flow chart - ocr.c



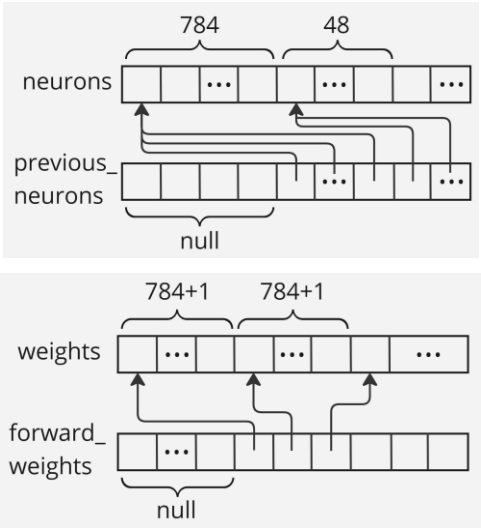
A. Data Structure in Neuronet

In `ocr.c`, it calls member function `neuronet_init` and `neuronet_load` provided by `NeuroNet` to initialize resources and to load all weights value.

There are 9 data members inside `NeuroNet`. `neurons` and `weights`, both of which are in the form of one-dimension arrays, are used to store value that will be used during evaluation. `previous_neurons` and `forward_weights` are array of pointers that points to some specific positions in `neurons` and `weights` respectively for easier data access. Their relationships are shown as Fig 2.

In this assignment, the MLP has 3 layers with 784, 48, and 10 neurons in each layer. Here in Fig 2, I directly use the number to distinguish different layers. `previous_neurons` of each neuron points to the head of previous layer. And know that the next layer of neuron n_l is calculate with the formula: $n_l = \langle W_{l-1}, n_{l-1} \rangle$. W_{l-1} is an array of weights with length as n_{l-1} plus addition bias 1. `forward_weights` of each neuron points to the head of W_{l-1}.

Fig. 2. Neurons, Weights, and related Structure



Another data member `output` point to the head of neurons in the output layer. The rest of 4 members are used to record the size of other data.

B. Mechanism of Evaluation

The whole workflow of `neuron_eval` can be summarize as following steps:

1. Copy the image into the 1st layer of `neruons`.
2. Iterate through all layers from 2nd layer.
3. Iterate through all neurons inside the current layer.
4. Calculate the value of current neuron.
5. Iterate through all neurons in output layer and return the maximum one.

In step 4, it first stores the pointers of `forward_weights` and `previous_neurons` of the current neuron in two variables `p_weight` and `p_neuron`. Know that the value of current neuron is calculated by using the formula $n_l = \langle W_{l-1}, n_{l-1} \rangle$ and both `p_weight` and `p_neuron` point to the head of W_{l-1} and n_{l-1}, there is an inner loop to iterate through whole data and perform inner product computation. This step is the key hotspot because of the computation and data feeding streams.

II. INSIGHT TO AQUILA AND IP

A. Interface between Aquila and SD card

To let Aquila able to communicate with SD card, there are two additional sources: `axi_quad_spi_0`, which is an IP used to interface SD card with the Aquila SoC through AXI bus; `core2axi_if`, which is responsible for converting the Aquila interface bus signal to the AXI-Lite bus signal for IP integration.

Observing the way `core2axi_if` interacting with slave device, we can find that there exist their own valid, ready, and data signal for different channels, where ready signal indicates that master is ready to receive output signal from slave and valid signal indicates that output signal from slave is able to use.

B. Aquila and IP

In `soc_top`, we can find there are a bunch of signals used to communicate with external device, and which device is chosen by using simple memory map. It is implemented based on target device address interval check.

From `core_top`, it issues a data request if the enable signal of write and read at execute stage is 1 and the data state is set to `d_IDLE`. At the same time, a target address is also sent out. Then in `aquila_top`, the address interval will be checked to determine whether to trigger an external device or not with strobe signal. If yes and there is a data request, then `M_DEVICE_strobe_o` becomes 1. Which device to choose is also determined by the address interval.

Signals like to transmit data from which device and if data from device is ready to send are also determined by checking memory address. During the execution of device, CPU stays in stall states until the requested data is ready.

III. IMPLEMENTATION

A. AXI FP IP Integration

Know that Aquila core can choose which external device to trigger based on the data address interval. We can utilize the feature to let FP IP read the correct input.

There are three input signals a, b, and c data in IP. They will be feed with `neurons`, `weights`, and `inner_product` respectively. In `neuronet.c`, declare 4 volatile pointers to specific address with 0xC4 at the beginning, three of them are used to store input values, while the last one is used to access the calculation result from IP that will be put in `inner_product` to calculate the next iteration.

To deal with input data and some trigger signals, there is an additional module `data_feeder` acts as a bridge with `soc_top` and FP IP. Because we know that CPU stalls after triggering the strobe signal, we only need to set the ready signal into 1 as finishing the calculation to let CPU keep executing from stall.

B. Data Storage - BRAM

We can find that whole `weights` are stored in DRAM by checking the data address. Know that BRAM has better performance than DRAM for data feeding and TCM is implemented with BRAM, I directly change the storage of whole `weights` from DRAM to BRAM. It is implemented by adding a new volatile pointer `nn_weights` with address in TCM to replace the original one.

C. Experiment

TABLE I. TIME COMSUME FOR EVALUATION

Modification		**Time (s)**	**Improvement (times)**
IP	BRAM		
		21.904	1
✔		4.602	4.76
✔	✔	2.946	7.43

We can find that there is a huge improvement between original Aquila multiplication mechanism and AXI FP IP that the execution time improve 4.76 times as long as original one. And, changing the data storage improves the time into 7.43 times than original one. Both implementations result in better performance.