COP820— Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks[1]

Devang Mahesh 2017EE10093
Ankit Garg 2017EE10439

I. Introduction

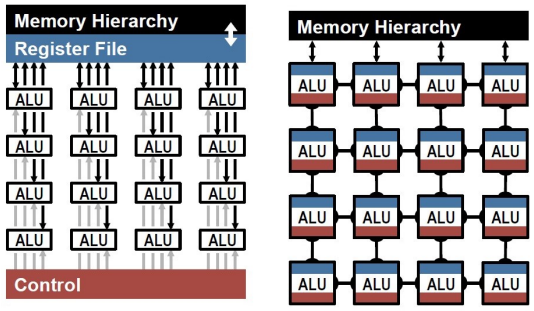
Eyeriss is an accelerator for state-of-the-art deep convolutional neural networks (CNNs). Modern Deep Learning models and systems are computationally heavy and running them on our normal system is quite inefficient in terms of power consumption. In this paper the authors tackle the problem by using a row stationary dataflow along with division of memory+control into different hierarchical levels.

In this course project we have replicated the dataflow and the parallelism methods proposed to get a functional block with all the steps right from loading the data from the Memory Banks to performing convolution to storing the Partial Sums back to the output Memory Bank.

II. Background

A. Architecture Proposed

The authors Propose to divide the Memory and the Controls more locally among each ALU which allow better reuse of data as shown in figure 1.



(a) SIMD Architecture (b) Spatial Architecture

Fig. 1: (a) The Generally Used Architecture (b) Proposed Architecture

Considering this as the base the figure 2 shows the complete architecture.
The main consideration are:

- The spatial architecture is implemented by using an array of 168 (12*14) processing elements (PEs)
- Each PE has local controls and a local memory called Spad. There are separate Spads for Input Feature Map, Output Feature Map and the calculated partial sums
- Due to the presence of Spads and inter-PE communication we save energy by not going to the Global Buffer and Dram to get data that in general consumes two to three order of magnitude of more power.

B. Dataflow Proposed

The Row Stationary Data Flow proposed uses various types of data reuse that minimise superfluous data movement to carry out convolutions that involves patterns that can be used by us for our benefit.

Three forms of data reuse have been used:

- **Convolutional Reuse:** Each filter weight is reused in a sliding fashion over the same ifmap plane E × F , and each ifmap pixel is usually reused R × S times in the same filter plane.
- **Filter Reuse:** Each filter weight is reused across the batch of N ifmaps
- **Ifmap Reuse:** Each ifmap pixel is reused across M filters (to generate M ofmap channels).

Where E X F are the 2D dimension of Output Feature Map, R X S are 2D the dimension of Input Feature Map and M is the number of filters used which is further equal to the number of Output Channels.

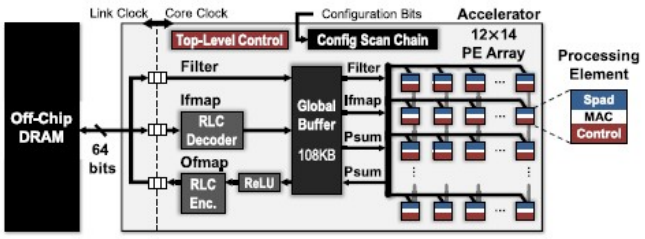


Fig. 2: Eyeriss System Architecture

Each PE performs 1-D convolution taking One row of ifmap and one row of fiter and then produce all the output partial sums **over time**. It can be seen in figure 3.

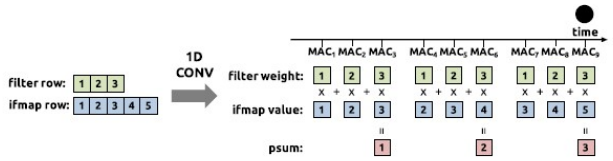


Fig. 3: Working of a PE

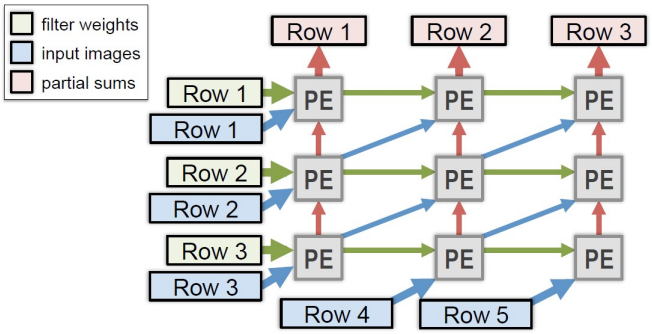


Fig. 4: DataFlow

Based on the functioning of a PE what we are left with is to divide the work among all the PE's. The final Row Stationary Dataflow has the following characteristics(Fig 4).

- The Filter rows are fed horizontally into the PE Array
- The Input Feature maps (ifmaps) are fed diagonally
- The Output partial sums are collected vertically after a vertical sum over the PE in a column

Building on this the authors propose interleaving of data to achieve minimum flow of data while calculation. Interleaving is to include the aforementioned a) Convolutional Reuse b) Filter Reuse c) Ifmap Reuse

- To include filter reuse the various ifmaps(of different input images) are interleaved sequentially i.e. one after the other. Fig 5a
- To include ifmap reuse the filters are interleaved in alternating fashion. Each ifmap data is worked upon by all the filters and hence keeping the ifmap value fixed all the filter values are passed. Fig 5b
- Different channels of same ifmap and filter can be interleaved. They are needed to be added to the same partial hence interleaving help in reduces the number of partial sums to be stores locally. Fig 5c

Till now we have looked at how work is divided among a group and within a PE. But if out Filter size is 3 then only three rows of the PE array will be used and other will be idling. To maximise the use we **divide PE array into PE sets** where each PE set takes some data in the **format as specified above Fig4 and interleaving Fig 5** . What they differ in is the data they get. If they get the different channels of identical ifmap and filter upward sum can be possible. If not different output partial sums are generated.

III. Symbols and Abbreviations

In a pass therefore:

Symbol/ Abbreviations	Description
S	The filter width
R	The filter height
ifmap	Input feature Map
psum	Partial Sums which are output of convolutions
p	The number of filters processed by a PE set. These no. of filters are interleaved while giving input to a PE set
q	The no. of channels of a particular filter processed by a PE set. Channels are interleaved after interleaving the 'p' filters.
r	No. of PE sets in the PE array that process different channels of a filter
t	No. of PE sets in the PE array that process different filters

1) p × t : Total number of filters passed to the PE array
2) q × r : Total number of channels of each filter passed to PE Array

IV. Implementation

(i) To set the re-configurable accelerator for a particular layer, we take the dimensions of the ifmaps, filters and the values of p,q,r,t as inputs. The master control takes these inputs and set the select lines accordingly.

(ii) **Memory banks:** The paper proposes to make the Global Buffer out of 25 banks,each of which is a 512-b ×64-b (4 kB) SRAM.
We have diluted this implementation and considered three banks of SRAM of appropiate sizes separately for ifmap,psums and filters. The proposed implementation of 25 banks requires us to have a tag for each data element. This was possible had we been implementing the Network on Chip for data movement. Since Network on chip has not been implemented we have taken some extra memory to make these separete banks.

The data is stored in the banks linearly i.e. one after the other. Each filter/ifmap is first changed from 3D to 1D and then stored into the bank. After this the second filter/ifmap will follow and so on. This can be seen in Fig 6.

The data can be accessed on a negative by giving the address and controling the write/read enable.

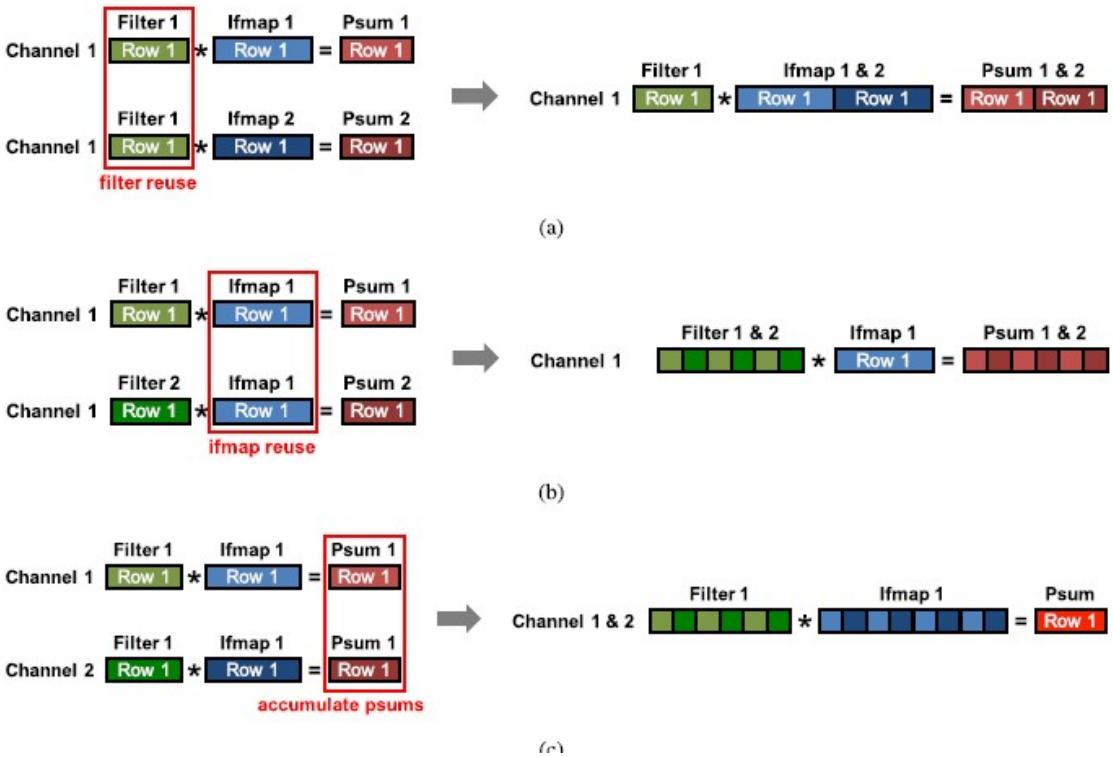


Fig. 5: Interleaving

(iii) **Master Control:** The master control co-ordinates the loading of ifmap and filter,execution followed by storing the output back to the banks.The master control generates the following signals:

- Enable Signals for the Banks: After the loading of the data from the DRAM to the Banks, the master control takes into consideration the values of p,q,r,t and then generates a read signal of sufficient duration for all the data to be transferred to the PE's.
- Load Signals: The point to note here is that all the PE set get different data. There is a common horizontal and diagonal meshes to feed data shown in Fig 4 as.Therefore we need control signals to decide whether a PE will take in data or not. Master Control generates these load signals.
  We need separate controls for filter and ifmap because in a pass all the filter data is passed together while the ifmaps are needed to be slided over time and to be given as input.
- MuX Controls: The partial sums are collected vertically and therefore output(opsum) of one PE is the input(ipsum) of the other. Since we divide PE array into PE sets, we need to break the upward sum chain according to the PE set division. We have placed MUX bw the output and input of every two PE. This controls whether the output from lower PE will go up or a Zero to break the chain will go up. See Fig 7

(iv) **Interleaving Control:** We need to interleave the data before passing it to the PE set. This is taken care by the loading module present along with the banks. They have counters that at each load enable change values and keep track of pass number representing the PE set to which the data is to be given.
Using the values of p,q,r,t the particular filters and their channels are selected for the PE set. Then the module interleaves them by incrementing local counters in a cyclic fashion.

(v) **Processing Element:** The processing elements is almost the same as proposed in the paper. Fig 8.Each computation pass is of 4 clock cycles. The scratch pads (spads) as were of size as indicated in the image and were implemented as SRAM. The other components were the two stage multiplier, the adder,the Muxes and PE control.

- Muxes: Two Muxes are shown in figure 8. The upper one is for controlling whether the result of multiplication or the ipsum will go in the adder. The control for this will depend on whether the computation of PE has been completed or not. While computation it sends the multiplication product while after computation it send the ipsum connected to output Mux vertically downward for upward sum.
  The lower PE is active during first 'p' calculation cycles when the psum present in the psum spad are garbage values. It sends zero during the initial 'p'

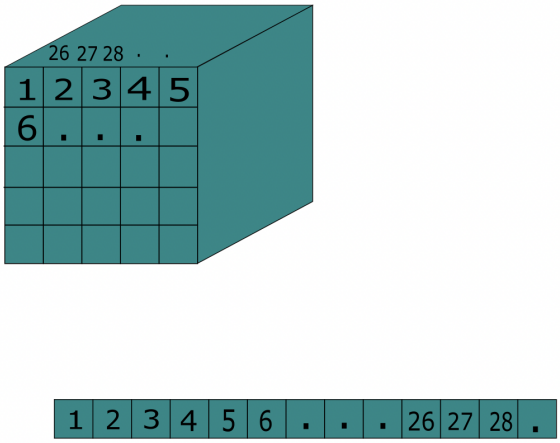


Fig. 6: Storing 3D Filters/Ifmaps Linearly

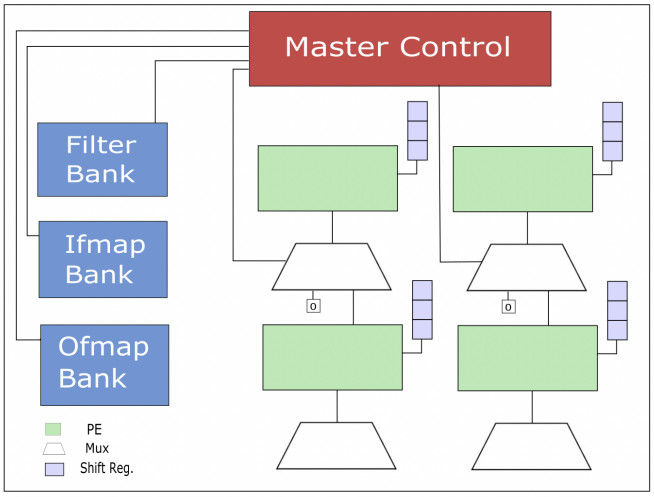


Fig. 7: Architecture Implemented

cycles. The controls for these are generated by the PE control.

- PE control: It generates the select signals for the MUXes using counters and the input values of p,q,S. It also uses the load signals from master control to set and increment the addresses of spads and to turn on the enables of the spads to load data. It also generates the write enable for the psum spad at every fourth clock cycle to store the calculates psum.

  It send a completion signal at the end of computation, back to the master control to load the next batch of ifmaps. (All PE have synchronous complete). It also generates the enable signals of the Shift register(will be explained next) so that the 'p' psums can be stored into the shift register after the computation is over.

(vi) Shift Registers: All the PE's generate Psums at the same time over 'p' clock cycles. The PE from which the data was to be taken is also dependent on r,t,p,q. This is too much data to be taken care of at the same time. So to reduce the traffics we have added shift registers next to every PE to store the output psums. All PE will generate psums but only the top PE row of every PE set will have correct values. All the PE store values to there shift register and then the output module generates the appropriate select lines to choose only the correct shift register to store results.

(vii) Output Module: It is not a separate module but present as a circuit within the PE array that takes the output from the correct shift registers, sequentially and places them in the output Memory Module at the appropriate address so that the final values in the Bank are output partial sum in their flattened form.

V. Evaluation

To check our implementation of the PE structure, we tested it on ifmap [1, 2, 3, 4, 5, 6,....26, 27, 28,....99, 100]. The

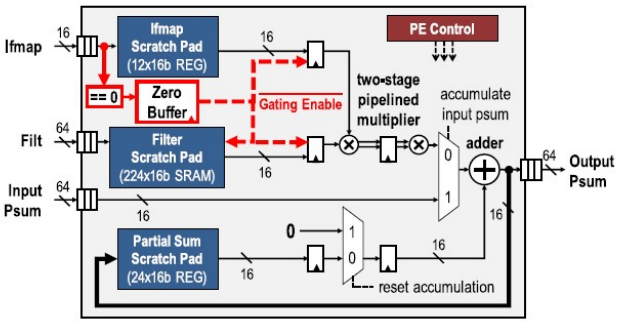
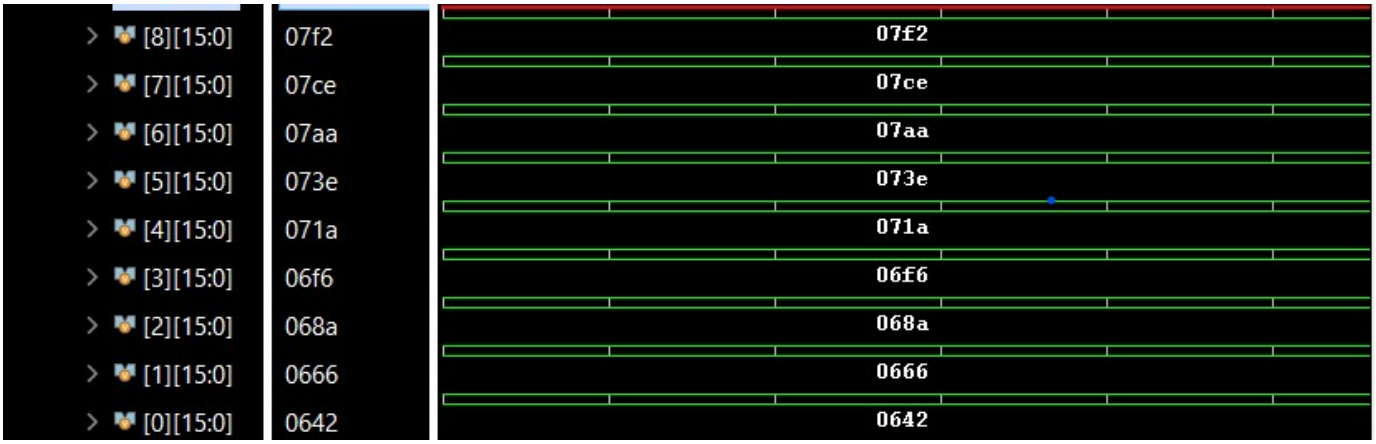


Fig. 8: PE Architecture

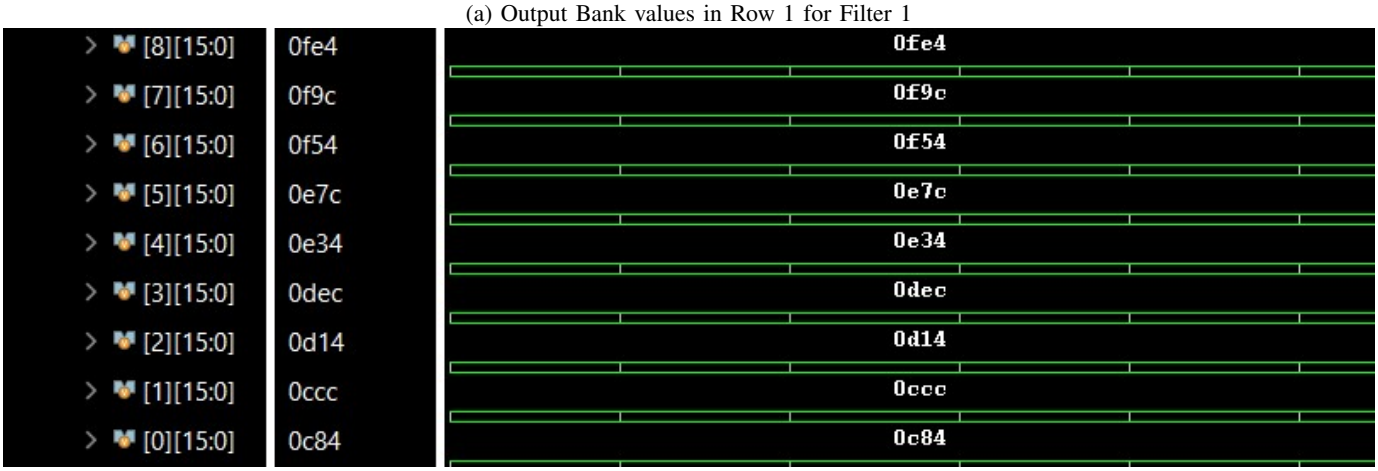
dimensions of ifmap are 5*5*4. We took two filters of 4 channels each. filter1 = [1,1,1 .... 9 times]. filter2 = [2,2,2 ... 9times], similarly filter3 and filter4. Their dimensions are 3*3*4. The output obtained were correct when calculated. The values are as given in the Table I. The Figure 9 represents the simulation results.

TABLE I: Parameters for testing.

Parameter	Value
H	5
W	5
S	3
R	3
p	2
q	2
r	2
t	2



(a) Output Bank values in Row 1 for Filter 1



(b) Output Bank values in Row 2 for Filter 2

Fig. 9: Simulation Results

VI. Future Work

Exploit zeroes in the ifmap to save processing power. We will use a 12 bit *Zero Buffer* to record the position of zeroes in the ifmap. We can avoid reading value from filter spad for zero ifmap value. We will try to come up with some more innovative ideas to save some clock cycles while loading from the filter anf ifmap banks.

References

[1] Y. Chen, T. Krishna, J. S. Emer and V. Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," in IEEE Journal of Solid-State Circuits, vol. 52, no. 1, pp. 127-138, Jan. 2017.