Deep Learning in Hardware
Prof. Naresh Shanbhag

Homework 4

ECE 498/598 (Fall 2020)
Assigned: Oct. 9 - Oct. 27

This is your last homework this semester and it contains 4 problems. Some questions and problems are marked with a star. These are intended for graduate students enrolled in the 598NSG section. Undergraduates enrolled in the 498NSU section are welcome to solve them for extra credit. The topics covered in this homework include: architectural mapping, importance of data reuse, systolic architectures, and algorithm transforms. This homework is due on Oct. 27, by 5pm, via Gradescope. No extension will be provided, so make sure to get started as soon as possible.

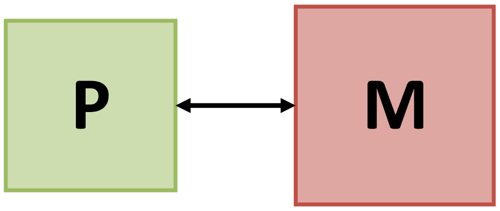


Figure 1: The considered hardware architecture in Problem 1.

Problem 1: Architectural Mapping Warm-up

In this problem, consider the simple compute architecture shown in Figure 2, constituting of a processor **P** and a memory **M**. The processor has a peak performance of 450 GMACs/sec with an arithmetic efficiency of 0.5 pJ/MAC. The memory read (write) bandwidth is 50 (20) Mbits/sec while consuming 10 (20) nJ/bit. The processor and memory are linked with a two-way communication bus, which is assumed to be ideal, that is, it has unlimited data transfer throughput and zero energy consumption.
The compute architecture is designed to process neural network layers in a layer by layer fashion. The processing of a layer is sequenced as follows:

- The layer weights and input activations are fetched from **M** to **P**
- **P** executes the computation required by the layer
- The resultant output activations are stored back in **M**

To simplify the problem, we assume that **P** can compute all types of neural network layers (CONV, FC) and that pooling/activation layers are performed automatically before the result is sent back to **M** at zero cost (latency or energy). Furthermore, it is assumed that the input activations can be discarded as soon as the output activations of that layer are written in **M**. Let's consider mapping AlexNet (see link: `https://github.com/pytorch/vision/tree/master/torchvision/models`, input size: $3 \times 224 \times 224$) onto this compute architecture.

1. Assuming all data types are quantized to 8b, what is the minimum allowable size of **M** (in bytes) needed to ensure that the architecture can perform inference without

any off-chip data access. Hint: you should consider both activations and weights.
Solution: **M** should have enough memory to store all the weights and the required activations. Since **P** processes the network in a layer by layer fashion, during the processing of each layer, we would need memory to store the inputs (which have to be read) and the outputs (which have to be written) for that layer. However, after a layer has been processed, its input activations need not remain in **M** and can be overwritten without any issues. Therefore the minimum allowable size of **M**:

$$\Big(\sum_{l=1}^{L} R_{W_l} B_{W_l}\Big) + \max_{l\in[L]}(R_{A_l}B_{A_l} + R_{A_{l+1}}B_{A_{l+1}})$$

where R_{W_l}, R_{A_l}, B_{W_l}, B_{A_l} are the number of weights, number of activations, precision of weights, precision of activations, at layer l, respectively. In order to map AlexNet onto this compute architecture, evaluating the above expression with the appropriate values yields a minimum allowable size of 58.46 MB.

2. Determine the total energy consumed by the compute architecture for a single inference. How much of the total energy is consumed by:

   - MAC operations
   - Memory read operations
   - Memory write operations

   Solution: The total energy consumed by this compute archtiecture for a single inference is:

$$E_{\text{inf}} = \sum_{l=1}^{L}\Big(N_l E_{\text{MAC}} + (R_{W_l}B_{W_l} + R_{A_l}B_{A_l})E_{\text{read}} + R_{A_{l+1}}B_{A_{l+1}}E_{\text{write}}\Big)$$

   where N_l is the number of MACs for layer l, $E_{\text{MAC}} = 0.5$ pJ, $E_{\text{read}} = 10$ nJ, and $E_{\text{write}} = 20$ nJ. Evaluating the expression using AlexNet numbers, we would get an inference energy $E_{\text{inf}} = 4949.58$ mJ. Out of which the total MAC energy is: 0.36 mJ, the total read energy is: 4916.48 mJ, and the total write energy is: 32.74 mJ.

3. Determine the inference latency by the compute architecture. How much of the total latency is due to:

   - MAC operations
   - Memory read operations
   - Memory write operations

   Solution: The total latency required by this compute archtiecture for a single inference is:

$$T_{\text{inf}} = \sum_{l=1}^{L}\Big(N_l T_{\text{MAC}} + (R_{W_l}B_{W_l} + R_{A_l}B_{A_l})T_{\text{read}} + R_{A_{l+1}}B_{A_{l+1}}T_{\text{write}}\Big)$$

where $T_{\text{MAC}} = \frac{10^{-9}}{450} = 2.22$ ps, $T_{\text{read}} = \frac{10^{-6}}{50} = 20$ ns, and $T_{\text{write}} = \frac{10^{-6}}{20} = 50$ ns. Evaluating the expression using AlexNet numbers, we would get an inference latency $T_{\text{inf}} = 9916.42$ ms. Out of which the total MAC latency is: 1.59 ms, the total read latency is: 9832.97 ms, and the total write latency is: 81.86 ms.

4. Assume now that a newer memory **M'** has been developed and will replace **M** in our compute architecture. Thanks to its smarter design **M'** can read and write $2\times$ faster than **M** with the same energy consumption. By how much will the inference latency change with the new memory block?
Solution: Using **M'**, the read and write latencies are halved. Therefore the new inference latency is $T_{\text{inf}} = 1.59 + \frac{9832.97+81.86}{2} = 4959.01$ ms, which results in an overall $1.999\times$ speedup.

5. If **M'** is instead $\beta\times$ faster than **M** with the same energy consumption. Plot the inference latency as a function of β. What do you observe?
Solution: For a compute architecture with a $\beta\times$ faster memory. The inference latency can be written:

$$T_{\text{inf}} = 1.59\text{ms} + \frac{9832.97 + 81.86}{\beta}\text{ms} \tag{1}$$

The inference latency vs. memory speedup is plotted below.

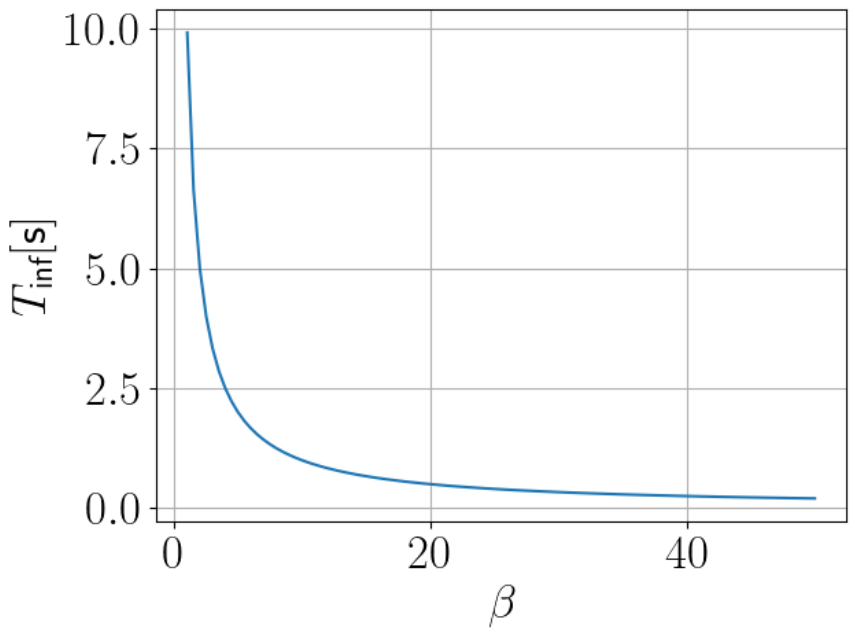


Figure 2: The considered hardware architecture in Problem 1.

Increasing β arbitrarily is meaningless beyond a certain point, as the minimum inference latency is lower bounded by the MAC latency which is 1.59 ms. This is referred to as Amdahl's law.

6. Plot the roofline and floorline plots for this architecture. Using the answer to Part 2, estimate the operational intensity (OI) for this workload.
Solution: The roofline plot is given in Fig. 3 and the floorline plot is given in Fig. 4. Since reading is the limiting factor, the slope should be the bandwidth for reading. Remember the slope should be converted to byte/s instead of bit/s.

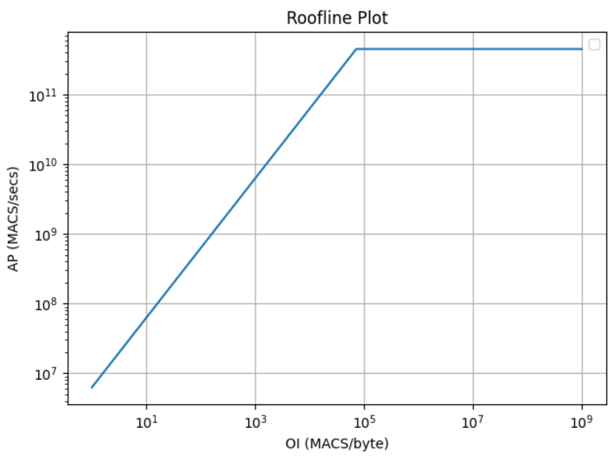


Figure 3: roofline plot.

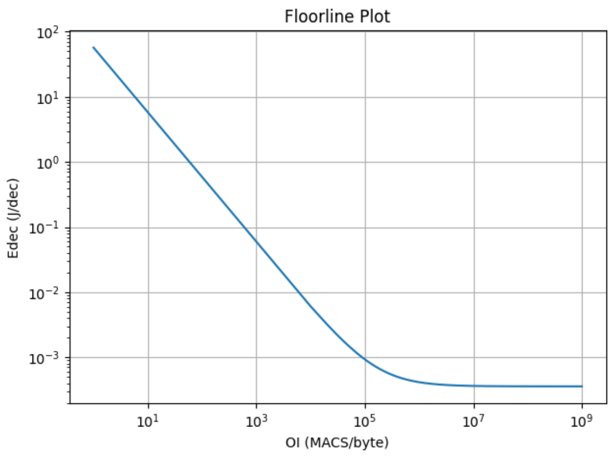


Figure 4: floorline plot.

Using the equation of the floorline plot, we can estimate the OI to be 11.54.

Problem 2: Importance of Data Reuse

In this problem, we will consider again the compute architecture presented in Figure 2. Some of the simplifying assumptions presented in Problem 1 for NN mapping will be removed. Let's consider the fifth convolutional layer (**C5**) in VGG-16 (see `https://github.com/pytorch/vision/tree/master/torchvision/models`), which comprises of 256 kernels, each of size $3 \times 3 \times 128$ (we will ignore the bias in this problem) and takes in input feature maps of size $56 \times 56 \times 128$. Unless specified otherwise, assume all datatypes are quantized to 8b.

1. Calculate the amount of data reuse for each of the input activations and weights. For a given datatype, the amount of data reuse is defined as the number of MACs that each data value is used for (i.e., MACs/data).
   Solution: **C5** takes in input feature maps of size $56 \times 56 \times 128$ and produces output feature maps of size $H_o \times W_o \times C_o$, where $C_o = 256$ is the number of convolution kernels, and $H_o = W_o = 56$ (zero padding with 2). Therefore **C5** requires $56 \times 56 \times 256 = 401408$ dot products, each of dimension $3 \times 3 \times 128 = 1152$. Thus the data reuse for weights:
   $$\frac{56 \times 56 \times 256 \times 128 \times 3 \times 3}{256 \times 128 \times 3 \times 3} = 56^2 = 3136$$
   Similarly, the data reuse for input activations:
   $$\frac{56 \times 56 \times 256 \times 128 \times 3 \times 3}{56 \times 56 \times 128} = 256 \times 3^2 = 2304$$

2. Assuming that there is zero input and weight data reuse in our architecture, that is, for every MAC operation that needs to be computed, both operands (weights and input activations) have to be read from the same external memory **M** (all partial sums generated by these operations are locally buffered and are readily available). If **P** has a peak MAC throughput of 450 GMACs/sec, what is the minimum required memory read bandwidth (bits/sec) needed to maximize utilization in **P** while executing **C5**?
   Solution: The peak MAC throughput of 450 GMACs/sec is achieved when all operands (inputs, weights, and psums) are readily available. Since psums are locally buffered, only inputs and weights have to be fetched form **M** for every MAC. Therefore the minimum required memory read bandwidth is $450 \times 2 \times 8 = 7200$ Gbits/sec, or 7.2 Tbits/sec, which is ridiculously high.

3. Assume now that **P** is equipped with an internal buffer that can store all the weights of **C5** for instant data access. The internal buffer allows us to reuse weights across different operations. What is the size (in bytes) needed for the internal buffer to fit the weights? What is the minimum required memory read bandwidth (bits/sec) needed to maximize utilization in **P** while executing **C5**?
   Solution: The internal buffer should have enough memory to store all the weights of **C5**, which amounts to $3 \times 3 \times 128 \times 256 = 288$ KB. Using the internal buffer, the minimum required bandwidth gets halved, because we only have to fetch one datatype (inputs) for every MAC from **M**. Therefore the required bandwidth becomes 3.6 Tbits/sec.

4. Mark the operating point of the architecture on the roofline plot of Problem 1 assuming: a) no data reuse as in Part 2, and b) with data reuse from Part 3.
Solution: We can see that when the BW (50Mbits/s) is less than the required band-

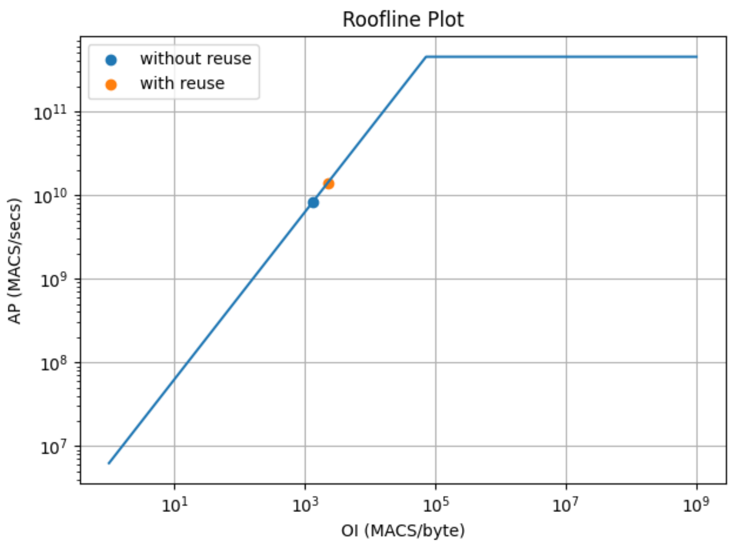


Figure 5: Marked Roofline Plot.

width calculated in part 2 and 3. The implementation is memory-bound (on the slant part of the graph). With reuse increasing the operation intensity (OI), the achievable performance (AP) also increases.

5. Would it be feasible to internally buffer input activations instead of weights? Briefly justify your answer.
Solution: Input activations for **C5** requires 392 KB of memory. Combined with the memory requirement for all weights, the total requirement is 680 KB, which makes it practically infeasible to store them in typically small buffer.

Problem 3: A 2D Systolic Architecture

Consider the 2D systolic architecture (SA) presented in Figure 6, which consists of:

- A mesh of 3×3 processing elements (PEs) **P0**,...,**P8**. Each PE contains its own local storage via a scratchpad which can be used to store **two data values** at a time, and can communicate with its neighboring PEs to send and receive one datatype each cycle (weights, inputs, or output partial sums). To be more precise, at each cycle, a single PE will first receive data from its neighbors (if any), perform one MAC operation, and send out one datatype to its neighbors (if required).

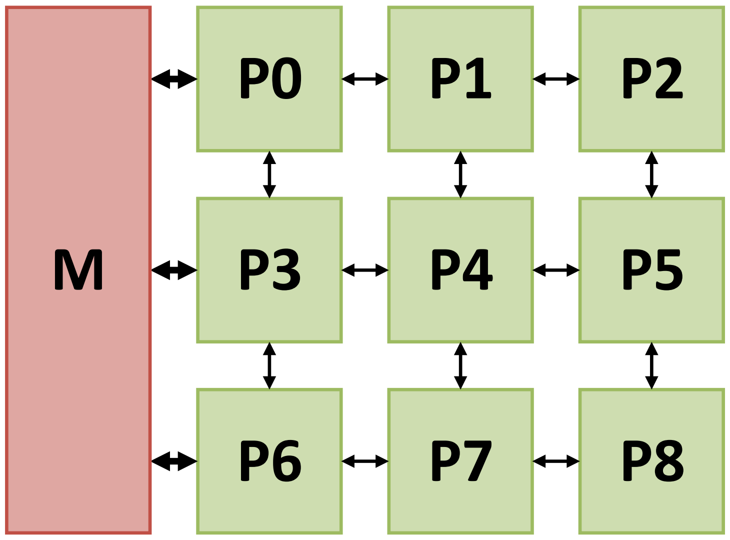


Figure 6: The considered hardware architecture in Problem 3.

- A main memory block **M** which is directly connected to **P0**, **P3**, and **P6**. **M** can also send and receive one datatype each cycle.

We would like to implement a simple 2D convolution using this proposed SA. The 2D convolution operates on a 5×5 input feature map, uses one 3×3 filter, and generates a 3×3 output feature map.
Note that output partial sums cannot be sent to **M**. In other words **M** can only store inputs, weights, and complete outputs. Furthermore, the execution of the convolution is considered complete when the last output is stored back into **M**. Assume that the contents of the PEs are *empty* before execution, and that **M** contains both the inputs and weights.

Three engineers proposed three different mappings listed below. For each mapping, Calculate the *number of cycles needed to finish execution*.

1. Mapping A: The first engineer noticed that the PE mesh has the same size as the output that needs to be computed, and proposed that each PE should process one output pixel locally. That is, partial sums are accumulated locally in each PE, and only weights/inputs are shared amongst PEs.
Solution: Mapping A is essentially an output stationary (OS) mapping. In OS, the processing of output pixels remains local in each PE as much as possible. In order for one PE to process one output pixel, it needs to perform $3 \times 3 = 9$ MACs using all 9 filter weights and a 3×3 patch of the input image. The 2D convolution example can

be written as:

$$\begin{bmatrix} y_0 & y_1 & y_2 \\ y_3 & y_4 & y_5 \\ y_6 & y_7 & y_8 \end{bmatrix} = \begin{bmatrix} w_0 & w_1 & w_2 \\ w_3 & w_4 & w_5 \\ w_6 & w_7 & w_8 \end{bmatrix} * \begin{bmatrix} x_0 & x_1 & x_2 & x_3 & x_4 \\ x_5 & x_6 & x_7 & x_8 & x_9 \\ x_{10} & x_{11} & x_{12} & x_{13} & x_{14} \\ x_{15} & x_{16} & x_{17} & x_{18} & x_{19} \\ x_{20} & x_{22} & x_{22} & x_{23} & x_{24} \end{bmatrix}$$

For example, computing pixel y_0 means performing $y_0 = w_0x_0 + w_1x_1 + w_2x_2 + w_3x_5 + w_4x_6 + w_5x_7 + w_6x_{10} + w_7x_{11} + w_8x_{12}$.

One solution for mapping A can be the following:

- Map each PE (Pi) to an output pixel y_i $\forall i \in \{0, ..., 8\}$.
- Perform processing over multiple passes, during each pass: the same weight is broadcasted (sequentially) to all PEs, and all required input pixels are streamed in to every PE.
- Once the right weight and input combination is present in each PE, it performs a MAC operation and accumulates the psum in its scratchpad.
- To finish execution, we need 9 such passes, one pass for every weight value.

We will explain the details of one pass using the following 3 steps:

1. The weight w_0 is sent to all PEs from **M**. This operation takes 3 cycles, as w_0 is propagated through P0, P3, and P6 from left to right. At the end of this step, each PE would have w_0 stored in its local scratchpad.
2. The first row of PEs P0, P1, P2 require input pixels x_0, x_1, x_2, respectively, whereas the second row P3, P4, P5 require input pixels x_5, x_6, x_7, respectively, and the last row P6, P7, P8 require input pixels x_{10}, x_{11}, x_{12}, respectively. Therefore, each row reads the required input pixels from left to right from **M** through P0, P3, and P6. This operation requires 3 cycles to finish, and at the begining of the third cycle each PE would have the same weight w_0 and the appropriate input pixel x_j.
3. During the third cycle from step 2: All PEs perform one MAC operation in parallel, and store their respective partial sum in their local scratchpad (since this is the first psum computed, accumulation is with zero).

Therefore, one pass requires $3 + 3 = 6$ cycles, and processes one weight pixel to completion. For the next pass w_1 is broadcasted in a similar fashion, and the required input rows change for each PE row (in reality they shift by one to the right). At the end of the last pass, each PE would be storing a complete output pixel, and an extra 3 cycles are needed to write them back into **M** (right to left). Thus, the OS mapping requires $(3+3) \times 9 + 3 = 57$ cycles to complete the execution of our workload. If the SA had a one cycle broadcast network on chip (NoC) connecting **M** to all PEs, the OS mapping can execute in fewer number of cycles.

2. Mapping B: The second engineer noticed that the PE mesh has the same size as the filter being used, and proposed that each PE should store one filter weight locally and share the other two datatypes (inputs and partial sums) amongst PEs.
Solution: Mapping B is essentially a weight stationary (WS) mapping. In WS, each PE would hold the same weight value during processing for as much time as possible. Using the same notation established above, one possible solution for the WS mapping is described as follows:

   - Each PE (Pi) stores one weight pixel w_i $\forall i \in \{0, ..., 8\}$ throughout processing.
   - Processing one output pixel to completion involves using all the weights, therefore the WS mapping processes the workload over multiple passes, where each pass computes one output pixel.
   - To finish execution, we need 9 such passes, one pass for every output pixel.

   For the WS mapping, there is a setup phase during which we first stream in the right weight value to the right PE. The setup phase requires 3 cycles to finish. During the first pass, we process the first output pixel y_0 as follows:

   1. Load the 3×3 input patch into the PE array. For y_0, the required patch is:

   $$\begin{bmatrix} x_0 & x_1 & x_2 \\ x_5 & x_6 & x_7 \\ x_{10} & x_{11} & x_{12} \end{bmatrix}$$

      This process requires 3 cycles, and during the third cycle, all PEs would have the required operands, and perform one MAC operation and store the resultant psums in their respective scratchpads.
   2. After step 1, every PE would be storing a psum $w_i x_j$. Computing y_0 requires accumulating all 9 psums before sending y_0 to $\mathbf{M}$. This can be easily performed in the PE array. The contents of the last row are sent to the middle row in parallel, while the second row accumulates the arriving psums with its respective psums. In the next cycle, the updated psums in the middle row are sent to the first row, while the first row accumulates the arriving psums with its respective psums. This process requires 2 cycles.
   3. After step 2, the PEs of the first row P0, P1, and P2 would each be storing the psums: $w_0x_0 + w_3x_5 + w_6x_{10}$, $w_1x_1 + w_4x_6 + w_7x_{11}$, and $w_2x_2 + w_5x_7 + w_8x_{12}$, respectively. Similar to step 2, we accumulate the psums across the PE array, this time from right to left. This requires 2 cycles and y_0 would be available in P0. An extra cycle is needed to send it back into $\mathbf{M}$.

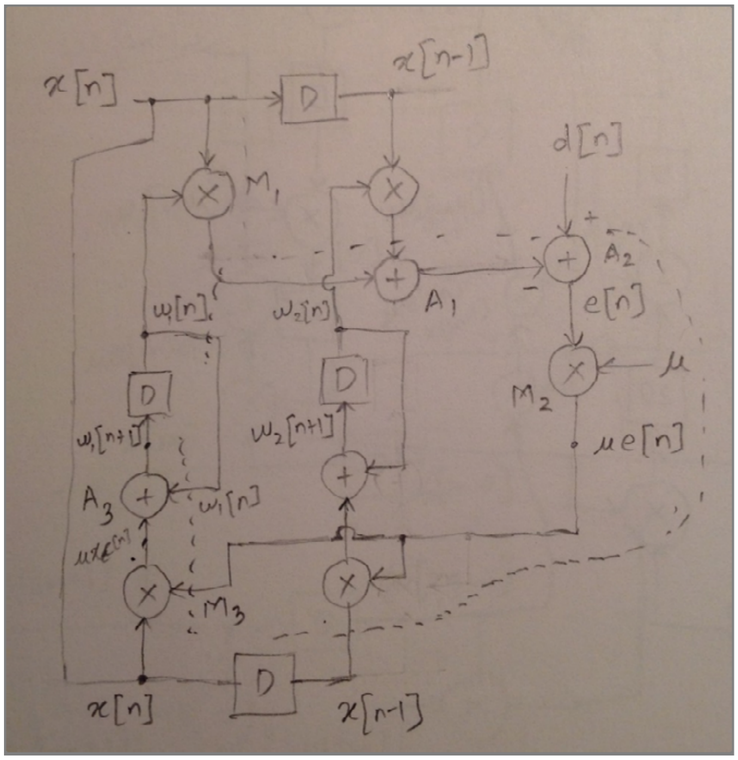
   Therefore, one pass requires $3 + 2 + 2 + 1 = 8$ cycles, and computing the workload requires $3 + 8 \times 9 = 75$ cycles, since 3 cycles are needed for setup and we need 9 passes to finish processing.

3. Mapping C: The third engineer was concerned about frequent fetches of input features from $\mathbf{M}$, and proposed loading a 3×3 patch of inputs onto the SA to process them completely. Once the corresponding output has been computed, another 3×3 input patch is sent again and the process is repeated.
   Solution: Mapping C describes an input stationary (IS) mapping. However the input shape is 5×5 and the PE array has a 3×3 shape, and since the problem requires that we only send complete output pixels to $\mathbf{M}$, then the 5×5 inputs have to be processed in patches of 3×3 across 9 passes (one for every output). Moreover, the processing of every output requires reading all 3×3 weight pixels for each pass. This mapping is essentially the same as mapping B, but inputs are fetched before weights are fetched. Therefore the same solution for Mapping B can be applied for C, and that would require 75 cycles.

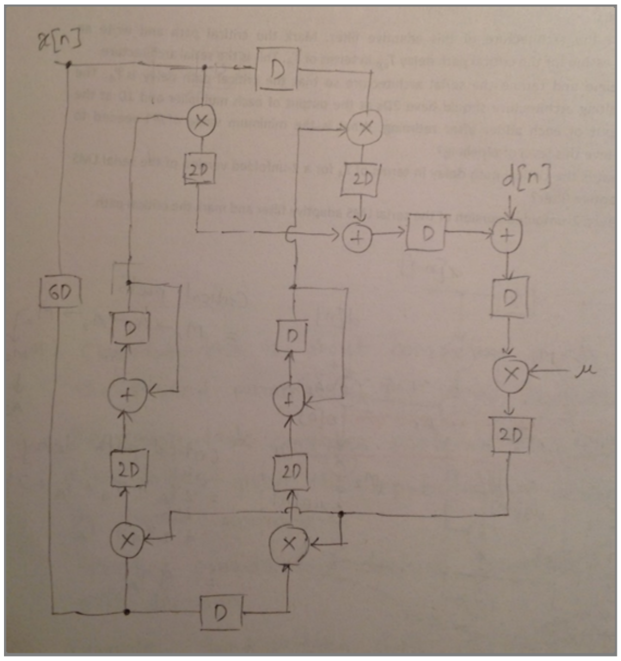
Problem 4: Algorithm Transforms *

Consider a 2-tap SGD-based adaptive time series predictor. Assume $T_M = 2T_A$, where T_M and T_A are the multiply and add delay times for all multipliers and adders (including the one in the WUD-block).

1. Draw the architecture of the SGD-based predictor. Mark the critical path and write an expression for the critical path delay T_{cp} in terms of T_A. This is the serial architecture.
   Solution: Critical path: $M_1 \rightarrow A_1 \rightarrow A_2 \rightarrow M_2 \rightarrow M_3 \rightarrow A_3$; $T_{cp} = 3T_M + 3T_A = 9T_A$.

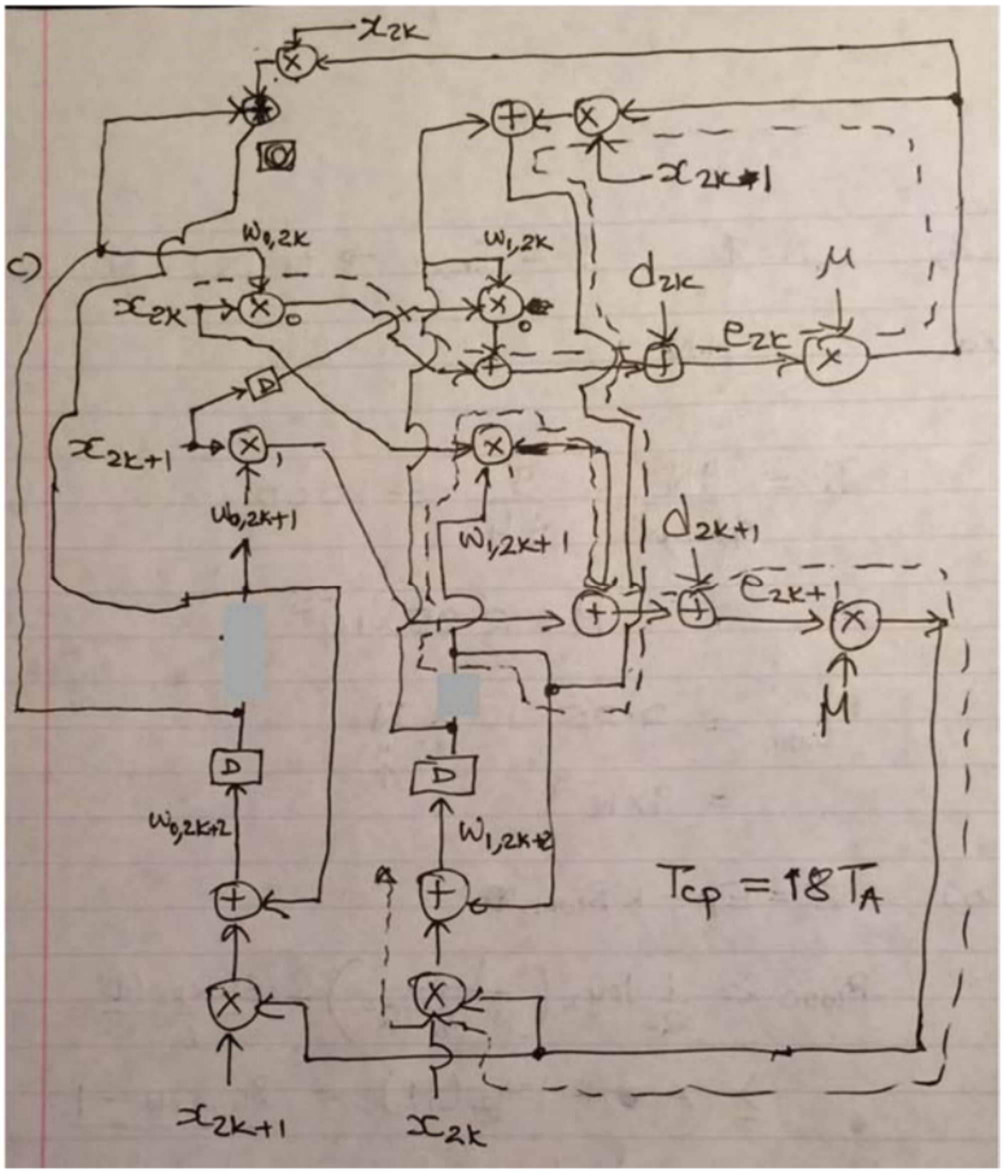


2. Pipeline and retime the serial architecture so that the critical path delay is T_A. The resulting architecture should have 2Ds at the output of each multiplier and 1D at the output of each adder after retiming. What is the minimum value of M_1 needed to achieve this level of pipelining?
Solution: The minimum value of $M_1 = 8$.



3. What is the critical path delay in terms of T_A for a 2-unfolded version of the serial architecture?
Solution: By definition of unfolding, a 2-unfolded architecture will have a critical path delay twice of the serial/original architecture. Hence, $T_{cp} = 18T_A$, for the unfolded architecture.

4. Draw a 2-unfolded version of the serial architecture and mark the critical path.
Solution: Please check the architecture below.


c)
x_{2k}
$w_{0,2k}$
$w_{1,2k}$
x_{2k+1}
d_{2k}
e_{2k}
μ
$w_{0,2k+1}$
$w_{1,2k+1}$
d_{2k+1}
e_{2k+1}
D
$w_{0,2k+2}$
$w_{1,2k+2}$
$T_{cp} = 18T_A$