EO 294: Systems for Machine Learning Assignment #3

Rohit R Jorige(22166)
Indian Institute of Science, Bangalore

March 6, 2025

Problem 1: Data Parallelism

a) Use 'all reduce' to update the gradients after every epoch.

Implemented and have attached codefile.

b) Use 'ring all reduce' to update the gradients after every epoch.

Implemented and have attached codefile.

c) Clearly describe how did you perform above two exercises. Compare the accuracy of the trained model and the training time.

We do the gradient synchronization (either All-Reduce or Ring All-Reduce) at a specific point in each training iteration, after the backward pass but before the optimization step.
All-Reduce
So first I create a zero tensor with the same shape as the parameter's gradient, then i accumulate all the gradients from all workers and divide by the number of workers to get the average gradient value. This is done by worker 0 and then passed onto all other workers.
Ring-all-reduce
I have broken this down into two phases (inspired by this article):

1. **Share-Reduce Phase:**
   - Each tensor is divided into p chunks (where p is the number of workers).
   - In each of the $p-1$ steps, each worker:
     - Sends a chunk to its neighbor in a ring topology.
     - Receives a chunk from its other neighbor.
     - Adds the received chunk to its corresponding chunk.
   - After this phase, each worker has one fully reduced chunk.
2. **Share-Only Phase:**

- In another $p-1$ steps, the fully reduced chunks are shared around the ring.
- Workers copy (without adding) the received chunks to build a complete tensor.
- After this phase, all workers have identical, fully reduced tensors.

For each parameter, the following steps were performed: First, the gradients were flattened into 1D tensors to facilitate chunking. Then, the Ring All-Reduce algorithm was applied. The results were averaged by dividing by the number of workers. Finally, the tensors were reshaped back to their original dimensions.

Multiprocessing:

I have used pytorch's multiprocessing library to simulate and coordinate multiple workers on my machine, but the gradient accumulation steps were not taken from the library.

The multiprocessing implementation works with these main parts. Worker processes are created using mp.Process in the main function, and model.share_memory shares the model. Each worker gets a unique rank, the shared model, and its own data portion. Data is split using np.array_split, and workers train on their subsets with Subset.

For inter-process communication, a gradients_queue (multiprocessing Queue) exchanges gradients, a results_queue gathers accuracy results, and a sync_barrier (multiprocessing Barrier) provides synchronization.

The synchronization workflow is as follows: workers compute gradients, send them to the queue, and wait at a barrier. Worker 0 (the coordinator) collects gradients and then applies All-Reduce or Ring All-Reduce, and sends reduced gradients back. Workers retrieve and apply these reduced gradients and continue on with the rest of the process.

Essentially, worker 0 acts as a coordinator, collecting gradients, applying synchronization, and distributing reduced gradients.

There are 4 workers spawned representing the 4 CPUs.

RESULTS: As we can see from the results obtained the accuracies are about the same

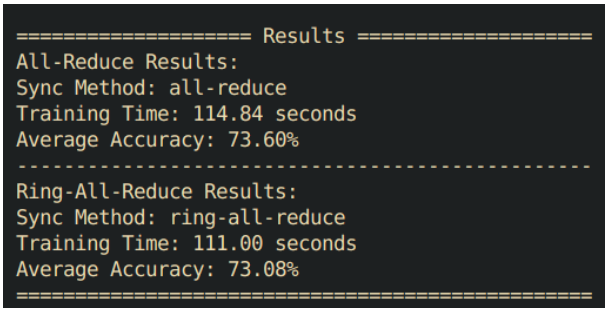

Figure 1: Results

while ring reduce has a slighty lower time taken due to less communication overhead.

Problem 3: Eyeriss row-stationary dataflow.

For this question I consider one cycle as MAC (the image on the left side) The green represents the elements of the filter , the blue represents the elements of the input feature map and the red represents the outpur feature map :

Below is the overview of each cycle:

The order in which the calculations occur is (PE1), (PE2), (PE3 , PE4) , psum(SUM(PE1 +PE2) , SUM(PE3 + PE4))

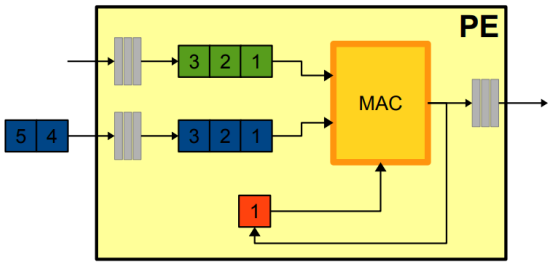

Figure 2: MAC Unit (Source: See Dataflow Lecture Notes)

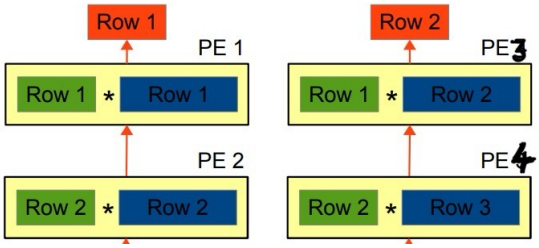

Figure 3: Processing Element (Source: See Dataflow Lecture Notes)

Source Link: `https://www.iitg.ac.in/johnjose/SPARC2/lecture3-dataflow.pdf`

For example, in the first cycle, a multiply-accumulate (MAC) operation is performed as follows:

$$1 \times (-1) + 3 \times 2 = -1 + 6 = 5$$

This MAC operation, as indicated by the MAC operator, is completed within a single cycle. In the second cycle, the MAC unit computes:

$$3 \times (-1) + 9 \times 2 = -3 + 18 = 15$$

These cycles continues in the order mentioned above and the results are calcuated in the figure below.

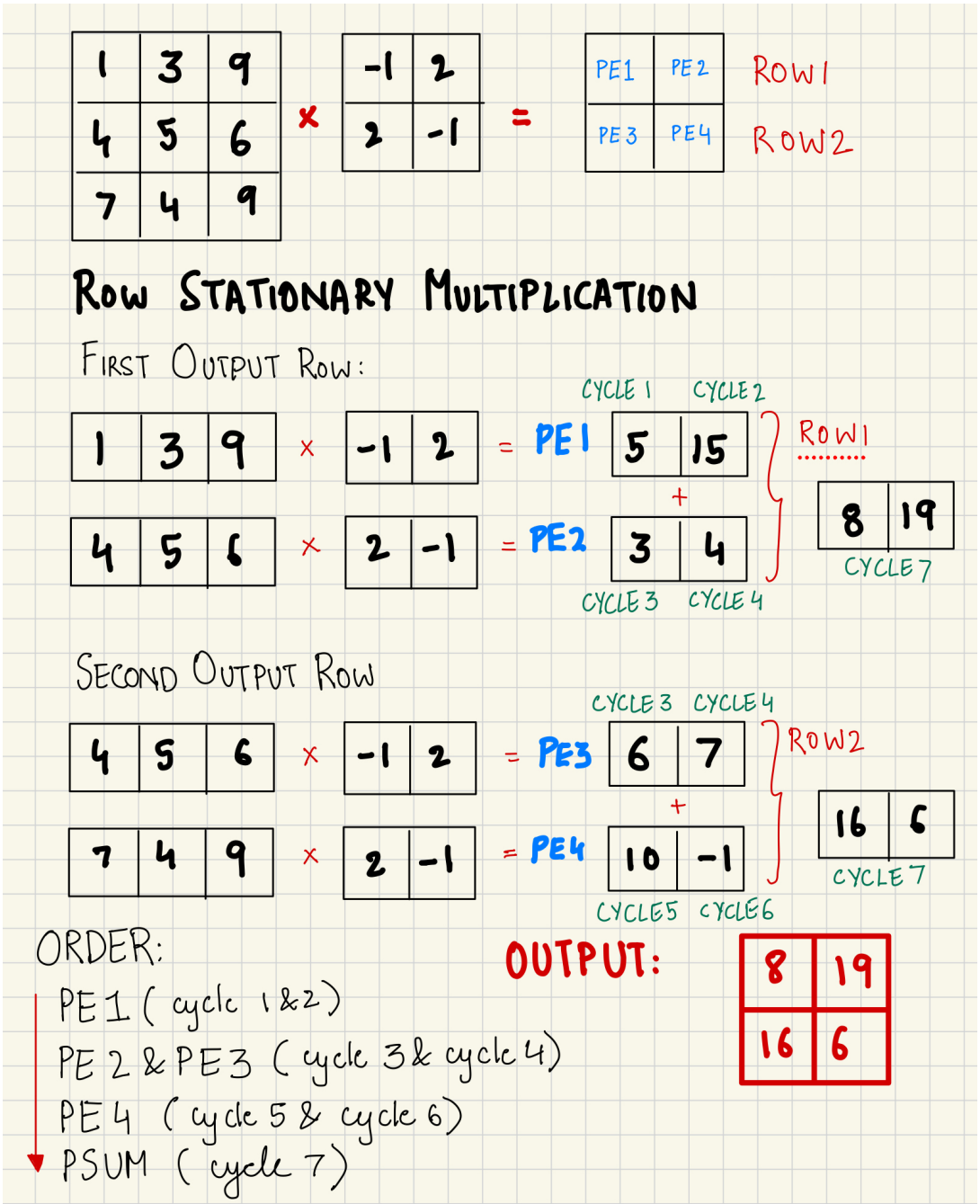

Figure 4: Working for Q3