CS348K: Visual Computing Systems
Class/Reading Response Template

Reminder: please make sure the PDF you submit to Gradescope DOES NOT have your name on it. We will concatenate all responses and give everyone in the class a PDF of all responses.

Part 1: Top N takeaways from discussions in the last class. Note: this part of the response is unrelated to the current reading, but should pertain to the discussion of the prior reading in class (or just discussion in the class in general, if there was no reading):

- What was the most surprising/interesting thing you learned?
    - I think the most interesting is dividing the matrices into smaller blocks to optimize GEMM. In this way, small matrices can be stored in cache memory, and the number of cache misses can be reduced. This also follows the philosophy of reordering.
- Is there anything you feel passionate about (agreed with, disagreed with?) that you want to react to?
    - Fuse can reduce memory access. Since intermediate results don't need to be stored in memory and are stored in registers instead, the access is much faster. Meanwhile, memory bandwidth can also be reduced. However, its difficult to image how fuse is implemented. In the TVM codebase, the fuse function is used to many places, e.g., combine the two axes, for each loop, scatter elements and so on. I think TVM combines two intermediate representations into one based on its graph templates. If there is a match, TVM will automatically perform the optimization.
- Did class cause you to do any additional reading on your own? If so, what did you learn?
    - I read more about convolution optimization and find that a commonly used one is Winograd convolution. Winograd convolution uses a set of pre-computed matrices, which are specific to the filter size and can be generated offline, which are then used to compute the convolved value by simple addition and multiplication. It reduces operations required for convolution and cuts down the computing time.
- Major takeaways in general?
    - Two optimization methods, blocking and fusion. One aims for cache and the other aims for memory.

Part 2: Answers/reactions to instructor's specific prompts for this reading. (Please see course website for prompts).

Like many computer architecture papers, the TPU paper includes a lot of facts about details of the system. I encourage you to understand these details, but try to look past all the complexity and try and look for the main lessons learned: things like motivation, key constraints, key principles in the design. Here are the questions I'd like to see you address.
What was the motivation for Google to seriously consider the use of a custom processor for accelerating DNN computations in their datacenters, as opposed to using CPUs or GPUs? (Section 2)

The motivation is for better cost and energy efficiency. In 2013 when a projection where people use voice search for 3 minutes a day using speech recognition DNNs would require Google datacenters to double to meet computation demands, which would be very expensive to satisfy with conventional CPUs. So they produce a custom ASIC for inference (and bought off-the-shelf GPUs for training) to improve cost-performance.

I'd like you to resummarize how the matrix_multiply operation works. More precisely, can you flesh out the details of how the TPU carries out the work described in this sentence at the bottom of page 3: "A matrix operation takes a variable-sized B*256 input, multiplies it by a 256x256 constant weight input, and produces a B*256 output, taking B pipelined cycles to complete”.

The matrix operation takes B pipelined cycles to complete, for each cycle, the MMU computes a 1 * 256 * 256 * 256 matrix multiplication, and uses a 4-stage pipeline to hide the execution of other instructions. The MMU uses a systolic data flow, the weights are preloaded, and take effect with the advancing wave alongside the first data of a new block. Control and data are pipelined to give the illusion that the input 1 * 256 is read at once, and that it instantly update one location of each of 256 accumulators, however, which is actually is the result of another input 1 * 256 several cycles ago.

We are going to talk about the "roofline" charts in Section 4 during class. These graphs plot the max performance of the chip (Y axis) given a program with an arithmetic intensity (X -- ratio of math operations to data access). How are these graphs used to assess the performance of the TPU and to characterize the workload run on the TPU?

Assess the performance: If the application can reach the ceiling (not slope), it achieves the best performance. We can decide whether it is under the slope, in which case it is memory bound, or under the ceiling, in which case it is computation bound.

Characterize the workload run on the TPU: If it is computation-bound, the workload is likely to to be CNNs.

Section 8 (Discussion) of this paper is an outstanding example of good architectural thinking. Make sure you understand the points in this section as we'll discuss a number of them in class. Particularly for us, what is the point of the bullet "Pitfall: Architects have neglected important NN tasks.”?

All nine papers looked at CNNs, and only two mentioned other NNs. Though CNNs are more complex than MLPs and prominent in NN competitions, they are only about 5% of Google datacenter NN workload. For us, I think the point is to focus on those are helpful and beneficial, not on those are hard and challenging. For engineers, valuable innovations may come from seemingly "dumb" ideas.

Part 3: [Optional] Questions I’d like to have specifically addressed via in class. (We also encourage you to just post these questions on Ed immediately so anyone can answer!)