ECE 498 ICC: IoT and Cognitive Computing

Spring 2020, Homework 2

Due in class on April 23, 2020

Question 1: IP-Based Hardware Design (10 pts)

Matrix-vector multiplications (MVM) are involved in most of deep neural network algorithms. Large MVM computations can be achieved by partitioning large inputs into small portions and feed into small MVM calculation IPs. Suppose we want to develop an IP-based hardware design for MVM.

1. List one advantage and one disadvantage for IP-based hardware design. **[2 pts]**
2. If we want to perform MVM on a matrix of size **2n**×**4n** and a vector with size **4n**, how many compute engine IPs do we need if:
   (a) the computer engine IP can calculate MVM with input sizes **0.25n**×**0.25n** and **0.25n**
   (b) the computer engine IP can calculate with input sizes **0.5n**×**0.5n** and **0.5n**

   You may assume that: we do not reuse IPs and there are enough buffers and adders to store and sum the intermediate results. **[3 pts]**
3. Consider IP version (b) from part (2) and assume we are allowed to reuse the IP. Each instance of the compute engine IP requires an area A and each use of an IP requires an additional $\frac{1}{4}A$ area. The latency of each IP to complete MVM on its input is T. You may assume that k IPs are instantiated, and they can work in parallel. However, each of them needs additional time, $\frac{k^2}{32}T$, due to data congestion. How many IPs should be instantiated to minimize the product of total latency and total area? Assume that the number of IPs is divisible by k.

   NOTE: You need to justify that the solution which you find is the global minimum of the objective function.

   Give reasons and calculate $Latency \times Area$. Suppose there is no pipelining, and all instantiated IPs run together. **[5 pts]**

Layers	Configuration	Activation Function	Output dimensions (channel, height, width)
convolution	5x5 filter, stride = 1, no padding	ReLU	(3, 24, 24)
max pooling	stride = 2	-	(3, 12, 12)
convolution	3x3 filter, stride = 1, with padding	ReLU	(3, 12, 12)
max pooling	stride = 2	-	(3, 6, 6)
fully connected	input size 108, output size 100	ReLU	(100, 1, 1)
fully connected	input size 100, output size 50	ReLU	(50, 1, 1)
fully connected	input size 50, output size 10	Softmax	(10, 1, 1)

Figure 1: Architecture of the FashionNet

Question 2: A Tiny CNN (25 pts)

Consider the CNN we used in Lab 1 as described in Figure 1:
The input image dimensions are (3, 28, 28). Assume there is no bias in any layers. You do not need to consider the activation layers either. The data type of all the numbers is **float**.

1. Write down the pseudo code for computing the first convolutional layer and the first fully connected layer. The input image is `img[3][28][28]`, the output feature map of the first convolutional layer is `conv1[3][24][24]`, and the convolutional filter is `conv_weight[3][3][5][5]`. The output from the second max-pooling layer is `pool2[3][6][6]`, which needs to be transformed into a 1D array as the input to the first fully-connected layer. The weights of the first fully-connected layer is `fc_weight[100][108]`, and the output of the first fully-connected layer is `fc1[100]`. **[5 pts]**

2. Calculate the total numbers of floating-point operations (FLOP) in both convolutional layers (2 convolutional layers) and all fully-connected layers (3 fully-connected layers) respectively. What is the total number of floating-point operations in this whole CNN (ignore the max pooling operations)? Compare the percentage of floating-point operations in convolutional layers and fully-connected layers in the whole CNN. **[4 pts]**

3. What's the total size of parameters in MB (megabytes) in the convolutional layers? How about the parameter size in all the fully-connected layers? What's the total size of parameters in this whole CNN? Compare the percentages of parameter size for those two parts of this CNN. **[4 pts]**

4. Calculate the computation-to-memory ratio (FLOP/MB) for the convolutional layers and fully-connected layers respectively. Compare these two ratios. **[4 pts]**

5. Based on the answers above, compare convolutional layers and fully-connected layers in terms of computation and memory. **[4 pts]**

Question 3: Batch Normalization (15 pts)

Batch normalization layers are used in most of today's state-of-the-art convolutional neural networks. Here is the paper https://arxiv.org/pdf/1502.03167.pdf, which introduces the batch normalization into the convolutional neural networks.

1. Write down the formulation of batch normalization layer. Describe the computations of batch normalization layer in your own words. **[5 pts]**

2. Explore the batch normalization API in TensorFlow. Consider the CNN we build in lab 1. Please write down the TensorFlow model (with Keras) of the first convolutional layer followed by a batch normalization layer (you don't need to write down the whole model for the CNN). **[5 pts]**

3. During the inference, the mini-batch mean and variance are fixed and independent of the input data. Consider fusing a convolutional layer and a batch normalization layer. Can you merge the batch normalization layer into the convolutional layer? **[5 pts]**

Question 4: GPU and CUDA (20 pts)

1. List at least three significant differences between GPU and CPU. **[4 pts]**

2. *Warp* is the basic unit in GPU. A warp is a set of parallel threads that execute the same instruction together in a Single Instruction Multiple Thread (SIMT) architecture. During execution, Streaming Multiprocessor (SM) does not touch individual threads but instead takes warps and executes them in a lock-step style.

   When threads in the same warp do "different things", we call it *control divergence*. As a good CUDA developer, you should try your best to minimize the control divergence. For this part, all warps have 32 threads linearly in the order of dimensions x >> y >> z.

(a) Consider the following code:

```
int thread_x = threadIdx.x;
int thread_y = threadIdx.y;
for (int i = 0; i < 16; i ++) {
    int pos_x = thread_x*16 + i;
    for (int j = 0; j < 32; j ++) {
        int pos_y = thread_y*32 + j;
        if (pos_x < 400 && pos_y < 384) {
            output[pos_x][pos_y] = 255 - input[pos_x][pos_y];
        }
    }
}
```

To revert the color of a grayscale image with a size of (400, 384) using GPU, you instantiate a kernel with a blockDim of (64, 16). Assuming each thread processes a sub-part of (16, 32), calculate the number of threads and warps that are in control divergence. Calculate how many wrap(s) are inactive? Justify you answer **[3 pts]**

(b) Now you are given the freedom to edit gridDim and blockDim. Please give a plan that can avoid any control divergences given the same task as in Part(a). You will need to give the configuration and a short code snippet on how to index the given grayscale image in your solution. **[3 pts]**

3. To evaluate the effectiveness of tiled matrix multiplication, you need to finish the following tasks. Calculate the total effective latency for the memory access of a regular matrix multiplication of two 4 by 4 matrices. Then, calculate the effective latency for a tiled matrix multiplication of two 4 by 4 matrices with TILE_WIDTH=2. Both latencies are measured in cycles. **[5 pts]**

   Please refer to lecture 10, slide 25 for the steps of calculating the latency. For latency of each read/write operation, please refer to slide 3.

   Memory access policy:

   (a) **No** concurrent access to the same memory location is allowed, such as global memory.

   (b) Concurrent access to **different** shared memory is allowed. **No** concurrent access to **same** shared memory is allowed.

   (c) When copying from the global memory to the shared memory, one action for copying data per tile pays the cost of only 1 read/write operation benefited from Burst Memory Access.

4. Given a system with one CPU and one GPU, you are asked to deploy a benchmark program to calibrate the performance of the GPU.

   (a) The manual of the single-core CPU tells you that each multiply-add operation requires 10 cycles to complete. Both GPU and CPU are running at the same clock frequency of 2GHz.

   (b) A part of the manual for the GPU is missing, but you are told that each multiply-add operation requires 250 cycles to complete. Each warp has 32 threads, and each Streaming Multiprocessor (SM) takes up to 1536 threads and spans no more than 8 blocks and 32 warps.

   (c) The benchmark program has the following characteristic:

   - Total multiply-add Operations: 5,120,000
   - The ratio of parallel and sequential code is 3 (75% – 25%)
   - Launch with a blockDim of ~~(64,3,1)~~ and a gridDim which is set for maximum possible concurrency (64, 2, 1)
   - All data on GPU has been preloaded

   You observe a Speedup of around **3.94225** against running on the same single-core CPU only. What is the number of Streaming Multiprocessor (SM) on this GPU? **[4 pts]**

Question 5: Processor Performance and Parallelism (20 pts)

You are asked to run an image recognition application on an edge device, and you need to decide what kind of processing device to use. Your initial test device is a single core processor running at 500MHz, and initial operating voltage is 1V. Your machine learning application has 100K instructions.

Part A:

When we profile a performance increase caused by running a portion of our code in parallel, we use Amdahl's Law to estimate the speedup:

$$Speedup = \frac{T_{new}}{T_{old}} = \frac{1}{(1 - Fraction_{parallel}) + \frac{Fraction_{parallel}}{Speedup_{parallel}}}$$

If you want to speed up your application by a factor of 10X, what percentage of your code should be parallelizable? **[2 pts]**

PART B:

To speed up your application, you may increase the core frequency, but this will increase the power consumption. Therefore, we explore the scenarios with multiple cores. Consider the following scenarios:

1. Single Core.
2. Four cores, and 10% of code is parallel.
3. Four cores, and 50% of code is parallel.
4. Four cores, and 90% of code is parallel.

You are given the following information:

1. Static power per core is 1W
2. Dynamic power per core is calculated as such:
   $P_{dynamic} = \alpha C V_{dd}^2 f$
3. Total power consumed per core is the sum of static and dynamic power
4. Each core has an average IPC (instructions per cycle) of 1.8
5. You may assume an activity factor $\alpha = 20\%$ and a total capacitance $C = 20nF$ (nano farad) per core.
6. When you scale up frequency, you must increase V_{dd} (voltage) as well. For every 500MHz above base frequency, you must increase V_{dd} by 0.15V. You may use the following formula:
   $V_{dd(new)} = V_{dd(base)} + 0.15 \times \lfloor \frac{f_{new} - f_{base}}{f_{base}} \rfloor$
7. When executing the serial (non-parallel) portion of code, only one core will consume dynamic power. All cores will consume static power.
8. For the fraction of code that runs in parallel, each core will run the same number of instructions. You may calculate the number of instructions in the parallel section based on the total number of instructions and the amount of parallelism available.

We will perform our analysis with the help of three metrics: Total latency (seconds), Peak Power (Watts), and Performance/Avg Power (MIPS/W), where MIPS = Million Instructions Per Second. Please report:

1. Create three plots for each metric, against increasing frequency. Limit the max frequency to 4.2GHz. **[8 pts]**

2. What is the average power, peak power, and latency in each scenario, at 4.2GHz? **[2 pts]**

3. At what frequency can the multi-core achieve similar Performance per Watt as a single core? What would happen if the static power per core was 2W? **[2 pts]**

4. If you want to use multi-core but limit peak power to 50W, how much parallelism do you need? **[2 pts]**

5. If you have a 50W average power budget, and code that is 90% parallel, should you choose a processor with 2-cores instead of 4-cores? What if your code is only 10% parallel? **[4 pts]**

Hints:

1. Please remember to check the units when computing each of the metrics.

2. Average power equals total energy divided by total latency.

3. Serial code is executed on a single core, and parallel code is executed on all cores.

4. When considering total energy and latency, you may want to consider serial and parallel portions separately.