Recitation 7 & Wet 1: Systolic Array Dataflows

EE 046853, Advanced Computer Architectures, Winter 2022/23
Technion — Israel Institute of Technology

1 Introduction

SCALE-Sim [1,2] enables research into CNN accelerator architecture and is also suitable for system-level studies. Since deep learning based solutions have become prevalent for computer vision over the recent few years, there has been a surge in accelerator design proposals both from academia and the industry.

It is natural to assume that we will see many more accelerators being proposed as new use cases are identified in the near future. However, it is important to keep in mind that CNN use cases will likely vary, which poses a large spectrum of efficiency and performance demands on the underlying CNN accelerator design. Therefore, it is important to quickly prototype architectural ideas and iterate over different designs.

In this document, we will go through SCALE-Sim, a CNN accelerator simulator that provides cycle-accuracy timing, memory bandwidth (BW) and trace results for a specified systolic array configuration and a neural network architecture. First, we will get to know SCALE-Sim, and see we understand its outputs. At the end of this document you will find your hands-on home exercise, which will also be graded.

1.1 Installation

SCALE-Sim is Python-based; therefore, no installation or compilation is required besides having Git, Python, and the relevant Python dependencies mentioned in the `requierments.txt` file. The lines of code below show how to clone SCALE-Sim Git repository and install the package dependencies with pip.

```
git clone https://github.com/ARM-software/SCALE-Sim.git
cd ./SCALE-Sim
pip install -r requirements.txt
```

If you do not have Linux installed or a remote access to a Linux machine, you can setup a virtual machine (using VirtualBox, for example), try use Cygwin or even Windows.

2 Getting Familiar with SCALE-Sim

SCALE-Sim is executed with the `python ./scale.py` command and requires two files: (1) the physical hardware configuration file (`--arch_config`); and (2) the network topology (`--network`), i.e., the number of layers and their dimensions.

2.1 Understanding Performance

To get familiar with SCALE-Sim, we start with a 1×1 systolic array in an output-stationary (OS) dataflow and a single convolution (CONV) layer. Configure `scale.cfg` and `test.csv` as described in Table 1. Then, execute the simulator with command below.

```
python ./scale.py --arch_config ./configs/scale.cfg
                  --network ./topologies/unit_tests/test.csv
```

1. **Q.** We observe 100% utilization — why? isn't there sparsity?
   A. SCALE-Sim does not assume anything about the dataset.

run_name	"SCALE_Sim_Example"
ArrayHeight	1
ArrayWidth	1
IfmapSramSz	128
FilterSramSz	128
OfmapSramSz	128
IfmapOffset	0
FilterOffset	10000000
OfmapOffset	20000000
Dataflow	os

Layer name	Conv5
IFMAP Height	8
IFMAP Width	8
Filter Height	3
Filter Width	3
Channels	32
Num Filter	100
Strides	1

Table 1: Initial configuration. (Left) hardware architecture. (Right) Network topology — in this case, a single CONV layer.

2. **Q.** The layer takes 1036801 cycles to compute — why?
A. Each ofm activation requires $3 \times 3 \times 32$ MAC operations; in addition, there are $6 \times 6 \times 100$ ofm activations. Therefore, the total MAC operations are 1036800. With an additional write cycle we get the same 1036801 cycles.

   In most cases,
$$\text{MAC Operations} = F_H \times F_W \times C \times O_H \times O_W \times F_N, \tag{1}$$
   where F_H, F_W, C, O_H, O_W, and F_N correspond to the filter height, filter width, channels, ofm height, ofm width, and the number of filters, respectively. As for the ofm dimensions:
$$\text{OFM Height} = \left\lfloor \frac{I_H - F_H}{S} + 1 \right\rfloor \qquad \text{OFM Width} = \left\lfloor \frac{I_W - F_W}{S} + 1 \right\rfloor, \tag{2}$$
   where I_H, I_W, and S are the ifm height, ifm width, and stride, respectively. Notice that the padding is already incorporated within the I_H dimension.

3. **Q.** Increasing the systolic array to 2×2 achieves the same utilization with quarter the cycles time. Does performance always increases linearly with the number of PEs?
A. That's part of your hands-on assignment.

4. **Q.** What is the minimal systolic array dimensions that can compute the entire ofm at once?
A. 36×100 (Figure 1). Execution performance is saturated at this point.

5. **Q.** SCALE-Sim evaluates the execution time as equals to 423 cycles — why?
A. In this case, the execution time is determined by the bottom-right PE. Only after data is propagated from the SRAMs through the PE array to the bottom-right PE, it can start performing actual MAC operations. That is, in this case, since the array accommodates the entire ofm,
$$\begin{aligned} \text{Execution Cycles} &= \text{SRAM Skew} + \text{Array Size} + \text{Computation} \\ &= (100 - 1) + (36 - 1) + 3 \times 3 \times 32 + 1 = 423. \end{aligned} \tag{3}$$

6. **Q.** A 36×100 systolic array achieves a utilization of 65% — why?
A. Each PE conducts exactly 288 MAC operations. During execution, however, each PE could have performed 424 MAC operations. Utilization is, therefore, at the 68% ballpark, as the simulator shows. In other words, there are relatively small number of computations compared to the computational potential of the large array.

7. **Q.** Assuming we can double layer dimension, which will increase the utilization the most?
A. The channel dimension. By doubling the number of channels we double the number of MAC operations per PE while less than doubling the execution cycles. Empirically, the simulation shows utilization of 80%. Analytically, each PE now conducts exactly 288×2 MAC operations; yet, each could have performed 711 operations. Therefore, utilization is around 80%. In other "words", the utilization, in this case, is
$$\frac{3 \times 3 \times C}{100 + 36 + 3 \times 3 \times C}, \tag{4}$$
   which converges to 1.

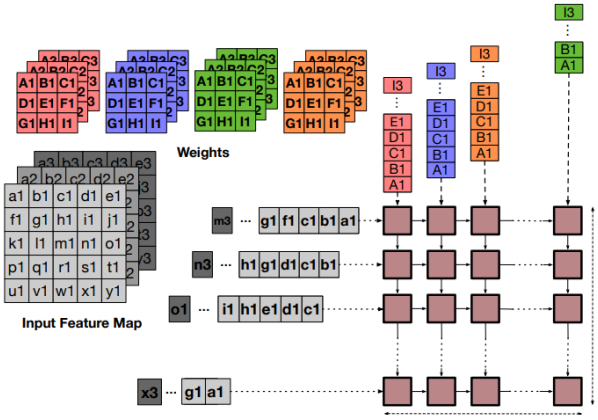

Figure 1: OS mapping schematic (from [2]).

2.2 Understanding Memory Accesses and Bandwidth

First, reconfigure the simulator according to Table 1.

2.2.1 DRAM Accesses

1. **Q.** The filter data is read from address 0 to 28799 — why? (the memory offset is omitted for readability).
   A. The filter size is $3 \times 3 \times 32 \times 100$. Notice it takes 2880 "negative" cycles, that is, cycles prior to the computation. Also notice that there are no additional accesses, so can empirically say that there is enough SRAM space to accommodate all weights.

2. **Q.** What is the smallest filter SRAM required to accommodate the entire weights at once?
   A. The filters require 28800B; therefore, 29KB. We can observe more DRAM accesses once the SRAM size goes below 29KB.

3. **Q.** The ifm data is read from address 0 to 2047 (neglecting the offset) — why?
   A. The ifm size is $8 \times 8 \times 32$. It takes 205 cycles to fetch them, in parallel to the filter accesses.

4. **Q.** The ofm data is read from address 0 to 3599 — why?
   A. The ofm size is $6 \times 6 \times 100$. It takes 359 cycles after the computation is over. The ofm activations can be saved during the computations, but the simulator is designed to save the ofm in bursts.

2.2.2 SRAM Accesses

1. **Q.** We observe sequential filter reads until 287 — why?
   A. This is exactly the number of weights, $3 \times 3 \times 32$.

2. **Q.** There is a sequential ifm read until 95, at which point there is a jump in the addresses — why?
   A. We can deduce that data is saved in C-W-H dimensions order. Therefore, there are 3×32 sequential reads and then a jump to the next line, which jumps across $(8-3) \times 32 = 160$ addresses. Indeed, the next access is to $160 + 96 = 256$. We can also observe the sliding window of the ifm and the repetitiveness of the filter accesses.

2.2.3 Bandwidth

Before checking out the BW readings, we will increase the systolic array computation capabilities to increase BW requirements. Set the systolic array dimensions to 36×100, and the SRAM sizes to 128KB.

1. **Q.** The ifm read BW is 0.56 B/cycle — why?
   A. Notice the average DRAM $\leftrightarrow$ SRAM BW is the number of total bytes versus the entire execution cycles, that is, including the ifm and weights fetch time and the ofmap save time. Therefore, $\frac{8 \times 8 \times 32}{2880+423+360}$.

2. **Q.** The filter read BW is 7.86 — why?
   A. $\frac{3\times3\times32\times100}{3663}$.

3. **Q.** The ofm save BW is 0.98 — why?
   A. $\frac{6\times6\times100}{3663}$.

The simulator does not consider stalls due to DRAM accesses, i.e., the bandwidth *requirements* fit the computational demands. Therefore, by decreasing the SRAM sizes we expect the *average* BW requirements to increase.

4. **Q.** Assume constant BW capabilities. Decreasing the SRAM size would, at some point, reach the BW limit which will decrease performance and will decrease the *average* bandwidth — how does it manifest in the Roofline model?
   A. First, the Roofline is a constant, since we do not change the peak performance or bandwidth capabilities. We do *decrease* the operational intensity, since the denominator, which is memory accesses in bytes, is increased.

3 Home Assignments

3.1 Instructions

1. Final submission deadline: 2/1/2023.
2. Submissions are in *couples*.
3. We will define the $MAGIC\_FILTER$ for each submission as follow:
   - Given a student id = 123456789, the 9^{th} digit equals to 9 and is denoted as ID_9^i when i is the student's number.
   - For a pair of students submitting together, calculate the $MAGIC\_SUM$ using both students' 9^{th} digits: $MAGIC\_SUM \equiv (ID_9^1 + ID_9^2) \mod 7$.
   - $MAGIC\_FILTER = MAGIC\_SUM + 1$.
4. Please submit your report to the appropriate homework submission box in Moodle as a PDF file.
5. Clearly explain your observations and insights. When presenting a graph, add axes labels and units. Feel free to ask questions in the appropriate Moodle forum. We encourage you to discuss all course material with your peers. However, the final solution must be yours, that is, a direct collaboration on assignments is strictly prohibited.
6. Due to the large period of time given for this exercise, no extra time will be given. Only official Technion reasons will be respected — reserve duty, hospitalization, or tragedy.

3.2 Initial Configuration

The homework assignment is based on the SCALE-Sim simulator [1, 2]. See the recitation for further details. Configure your simulation using the following instructions:

- Configure a 16×16 output stationary (OS) systolic array with 128KB SRAMs.
- Create a copy of the MobileNet topology [3] from this path ./topologies/conv_nets/mobilenet.csv
- Change all layers with filter size 3×3 to filter size $MAGIC\_FILTER \times MAGIC\_FILTER$.

Execute your modified MobileNet topology with the command line below (set the paths to your configuration file and topolgy).

```
./scale.py --arch_config [YOUR_CONFIG_FILE]
           --network [YOUR_TOPOLOGY_FILE]
```

3.3 Questions

1. Present a single bar plot of the utilization for the different layers.
   (a) Explain the source for the severe underutilization of some of the layers.
   (b) The underutilized layers were given a name. What is it? What is the essence of these layers? Find it in the MobileNet paper [3].
   (c) Explain what depthwise separable convolutions are, and the motivation behind them. Again, use the MobileNet paper [3].
2. Run the same topology with weight stationary (WS) and input stationary (IS) dataflows.
   (a) Present a single bar plot of the utilization of the different layers for the three dataflows.
      i. If there are differences between the dataflows, explain why. For example, the IS and WS utilization of layer i are different — why?
      ii. If there are no differences between the dataflows, explain why. For example, the IS and WS utilization of layer i are pretty much the same — why?
   (b) Present a single bar plot of the execution cycles of the different layers for the three dataflows.

i. What is the IS and WS dataflow *speedups* over the OS dataflow in layer granularity (i.e., we assume the OS dataflow is the baseline). Also, what is the *overall* speedups of the IS and WS dataflows over the OS dataflow.

ii. Something weird about the utilization and cycles? (Hint: if utilization is increased do we expect an increase or a decrease in cycles?). How do you settle this? (Hint: read "Runtime with limited MAC units", page 6 in [2] and check the SRAM traces).

3. Explore the impact of different systolic array dimensions on the overall execution cycles. Use the following dimensions: 32×512, 64×256, 128×128, 256×64, and 512×32. You can add more if you'd like, just notice the number of PEs is a constant.

   (a) Present three scatter plots with straight lines and markers, one for each dataflow, that present the execution runtime in cycles versus the systolic array dimensions (dimensions order as specified above). Explain the different trends.

4. Let's incorporate the Roofline model. Reconfigure your systolic array to an OS 16×16 with 4MB SRAMs (an ideal SRAMs assumption).

   (a) Present a Roofline plot (without the Roofline) with points that represent each layer operational intensity. Do not forget to set the axes to log2-scale.

   (b) On top of (a) draw the Roofline. The performance hard coded in the configuration file. The BW, however, is not. What is the minimal BW needed so no layer reaches the system BW cap? Use the Roofline to get the number, so we do not expect a super accurate number.

   (c) Looking at the Roofline, what can you say about your systolic array? Is it well suited for this neural network topology?

   (d) Repeat (a) and (b) with a systolic array with 64KB SRAMs.

   (e) Did the layer points moved horizontally? vertically? explain.

References

[1] Samajdar, A., Zhu, Y., Whatmough, P., Mattina, M. and Krishna, T., 2018. "SCALE-Sim: Systolic CNN accelerator simulator." *arXiv preprint arXiv:1811.02883*.

[2] Samajdar, A., Joseph, J.M., Zhu, Y., Whatmough, P., Mattina, M. and Krishna, T., 2020. "A systematic methodology for characterizing scalability of DNN accelerators using SCALE-Sim." *In Symposium on Performance Analysis of Systems and Software (ISPASS)* (pp. 58-68). IEEE.

[3] Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M. and Adam, H., 2017. MobileNets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.