Embedded System Design and Modeling

EE382N.23, Fall 2019

Lab #1

Specification

Due: September 30, 2019 (11:59pm)

Instructions:

- Please submit your solutions via Canvas. Submissions should include a lab report (single PDF) and a single Zip or Tar archive with the source and supplementary files (code should include a README and has to be compilable and simulatable by running 'make' and ‘make test’, respectively).
- You are allowed to work in teams of up to three people and you are free to switch partners between labs and the project. Please submit one solution per team.

Tiny YOLO Object Detector Specification Model

The purpose of this lab is to convert the TinyYOLOv2 object detection neural network into a clean specification model that conforms to the KPN model of computation (MoC) discussed in class. TinyYOLOv2 is a smaller version of YOLOv2 that targets resource-constrained environments. It consists of 16 layers, out of which 9 are convolutional. You can find more details about in [1]. Although YOLO/TinyYOLO has been implemented in various frameworks, for purposes of this course, we will use the original C implementation in Darknet [2].

You can install Darknet in Linux using commands below:

```
% git clone https://github.com/pjreddie/darknet
% cd darknet
% make
```

Next, we will need to download pre-trained weights of TinyYOLO:

```
% wget https://pjreddie.com/media/files/yolov2-tiny.weights
```

Now we can use command below to detect objects in sample images provided by Darknet:

```
% ./darknet detect cfg/yolov2-tiny.cfg ./yolov2-tiny.weights data/dog.jpg
```

If everything works, you should get an output similar to Figure 1. As this figure shows, TinyYOLO predicts three objects in this picture. You can check the location and size of predictions in `predictions.jpg` that is generated by Darknet automatically when executing command above.

```
osboxes@osboxes:~/Documents/darknet$ ./darknet detect cfg/yolov2-tiny.cfg ../darknet_weights/yolov2-tiny.weights data/dog.jpg
layer     filters    size              input                output
    0 conv     16  3 x 3 / 1   416 x 416 x   3   ->   416 x 416 x  16  0.150 BFLOPs
    1 max          2 x 2 / 2   416 x 416 x  16   ->   208 x 208 x  16
    2 conv     32  3 x 3 / 1   208 x 208 x  16   ->   208 x 208 x  32  0.399 BFLOPs
    3 max          2 x 2 / 2   208 x 208 x  32   ->   104 x 104 x  32
    4 conv     64  3 x 3 / 1   104 x 104 x  32   ->   104 x 104 x  64  0.399 BFLOPs
    5 max          2 x 2 / 2   104 x 104 x  64   ->    52 x  52 x  64
    6 conv    128  3 x 3 / 1    52 x  52 x  64   ->    52 x  52 x 128  0.399 BFLOPs
    7 max          2 x 2 / 2    52 x  52 x 128   ->    26 x  26 x 128
    8 conv    256  3 x 3 / 1    26 x  26 x 128   ->    26 x  26 x 256  0.399 BFLOPs
    9 max          2 x 2 / 2    26 x  26 x 256   ->    13 x  13 x 256
   10 conv    512  3 x 3 / 1    13 x  13 x 256   ->    13 x  13 x 512  0.399 BFLOPs
   11 max          2 x 2 / 1    13 x  13 x 512   ->    13 x  13 x 512
   12 conv   1024  3 x 3 / 1    13 x  13 x 512   ->    13 x  13 x1024  1.595 BFLOPs
   13 conv    512  3 x 3 / 1    13 x  13 x1024   ->    13 x  13 x 512  1.595 BFLOPs
   14 conv    425  1 x 1 / 1    13 x  13 x 512   ->    13 x  13 x 425  0.074 BFLOPs
   15 detection
mask_scale: Using default '1.000000'
Loading weights from ../darknet_weights/yolov2-tiny.weights...Done!
data/dog.jpg: Predicted in 1.941795 seconds.
dog: 82%
car: 74%
bicycle: 59%
```

Figure 1. Output of TinyYOLO-Darknet when run with dog.jpg

We will use the SystemC C++ class library [5] to model TinyYOLO as KPN. SystemC provides an easy-to-use threading interface and thread-safe, race-free communication and FIFO queue semantics. Note that SystemC has a lot more to offer and is an industry-standard system-level design language (SLDL) used for system design and modeling, especially at the system-on-chip (SoC) and hardware level. We will learn and use more of SystemC for SoC and hardware/software implementation modeling later in class.

Follow the tutorial posted on the class webpage [3] to setup the SystemC environment and become familiar with it. We have created a template library for modeling of KPNs in SystemC, available at [4]. Follow the tutorial provided with the library, and study and run the provided KPN examples.

It is then time to parallelize and refine the Darknet/TinyYOLO code into a KPN.

(a) Layer-wise pipelining

One of the most straightforward ways to refine TinyYOLO's neural network is to model each layer as a separate Kahn process. Implement such refinement by defining classes that inherit from `kahn_process` defined in [4], define ports necessary for interfacing it to other layers and override its `process()`with the neural network layers' forward propagation method. Given that TinyYOLOv2 has 9 convolutional and 6 maxpool layers, it is highly recommended that you develop a single parameterized convolutional and a single parameterized maxpool layer process template to implement those layers in TinyYOLO. Note that, besides convolutional and maxpool layers, TinyYOLOv2 has a region layer which you will need to implement as well.

Finally, in addition to the layers in the neural network itself, you will also need to model a testbench of pre- and post-processing processes and interface them to the neural network model. Pre-processing consists of loading input image from a file and adjusting its size. Post-processing includes non-maximal suppression (nms) that eliminates duplicate predictions and writing predictions to a file. See the hints at the end of this document for more details.

Make sure that your final model compiles, simulates and produces the same golden reference output as the original code. Document your model in terms of processes and their connections, e.g. in graphical form. Profile the model in terms of the computation, memory and communication requirements of each process. You can profile computation requirements using '`gprof`' (make sure to compile the code with the `-pg` flag), including a breakdown of the time spent in different methods or functions called by a process. Memory and communication requirements can be determined by analyzing (or instrumenting) the code. For memory footprints, break them down in terms feature maps and weights.

(b) Tiling and fusing

A layer-based partitioning results in large feature maps exchanged between layers. As a next step, further explore alternative ways of modeling the neural network that expose additional parallelism while reducing the communication and memory requirements per process. One way to achieve this is by exploiting inherent locality between input and output feature maps of a convolutional layer to partition feature maps into smaller tiles that can be computed independently and concurrently. To further reduce communication, matching tiles across layers can then be fused together into a larger process.

Such a fused tile partitioning (FTP) approach is described in [6]. Refer to [6] to first partition TinyYOLO's layers 0-14 each into nine processes using a 3x3 grid of equally sized tiles. What are the computation, memory and communication requirements of each process now?

Then further fuse matching tiles of layers 0-14 into nine independent Kahn processes according to the full FTP approach described in [6] (to apply FTP, you will need tile coordinates for input/output feature map of all layers 0-14, which you can derive using Algorithm 1). What are the per-process computation, memory and communication requirements after fusing?

To reduce the programming effort, it is recommended that you define a single parameterized Kahn process for tiled and fused layers that takes input and output coordinates as parameters. Note that region layers, as well as pre/post processing, cannot be tiled/fused. As such, they are not included in the fused layers and should be modeled as standalone processes similar to those in part (a). Furthermore, in defining ports and connections, you should pay attention that some of these standalone processes might need more than one output port or more than one input port.

Additional Questions

Answer the following additional questions:

(a) Is it always optimal to fuse all layers in FTP?

(b) Can your different KPNs run in bounded queue memory? If not, why not? If yes, what are the possible schedules and the minimal queue sizes that guarantee model operation, and what effect does increasing the queue size have on the model and its potential implementation?

(c) Can the different partitioning schemes be modeled using a SDF MoC? If so, describe how you would change `kahn_process.h`. If not, explain why, i.e. what are the limitations of SDF that prevent modeling the application as SDF.

Hints

- Keep in mind that in a KPN, memory/state can only exist within processes, i.e. there are no global variables, and all communication between processes must be through FIFO queues.
- To understand Darknet's code flow, checkout examples/detector.c::test_detector()
- Use Darknet's load_image_color() and letterbox_image() to load input image and interface it to the neural network.
- You can find layer implementations in the "src" folder of Darknet.
- Note that maxpool layers are stateless but convolutional layers have weights/biases that need to be loaded from weights file. For more details on loading layer parameters, see src/parser.c::load_convolutional_weights().
- Use draw_detections() to generate images with predictions marked on the original image.

References

[1] J. Redmon and A. Farhadi. "YOLO9000: better, faster, stronger," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[2] https://pjreddie.com/darknet/YOLOv2/

[3] http://www.ece.utexas.edu/~gerstl/ee382n_f19/docs/SystemC_setup.pdf

[4] https://github.com/kammirzazad/KPN-SystemC

[5] IEEE Standard SystemC Reference Manual (IEEE 16666-2011), https://standards.ieee.org/standard/1666-2011.html

[6] Z. Zhao, K. Mirzazad Barijough, and A. Gerstlauer. "DeepThings: Distributed Adaptive Deep Learning Inference on Resource-constrained IoT Edge Clusters," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37.11 (2018): 2348-2359.