Tutorial of the simulator for TPU-like system

Developers: Anni Lu, Junmo Lee, and Yandong Luo
PI: Prof. Shimeng Yu, Georgia Institute of Technology
November 12th, 2024

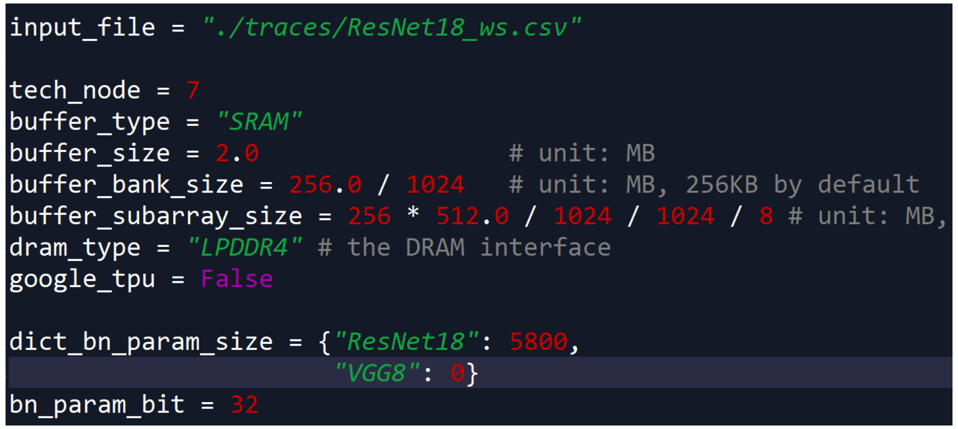
Overview

- The simulator was developed by Anni Lu, Junmo Lee, and Yandong Luo to evaluate the performance of TPU-like systolic array using eNVM device technologies as the on-chip global buffer for the paper *High-speed emerging memories for AI hardware accelerators.*
- It is written by python. So, please be familiar with python if you want to customize it for your own work.
- The simulator reads the traces from an auto-mapper time-loop (by MIT) to get the number of mac operations and memory accesses at each level of memory hierarchy. To make it simple, traces has been provided for VGG-8 (CIFAR-10 dataset) and ResNet-18 (ImageNet dataset) under the folder ./traces
- The hardware performance is evaluated using the circuit-level modules from NeuroSim. Currently, it supports different memory technologies such as SRAM, eDRAM and eNVMs. The users needs to provide the area, read/write energy as the input.
- Please cite the following references if you use it for your work:
    - A. Lu, J. Lee, T. H. Kim, M. Karim, R. S. Park, H. Simka, S. Yu, High-speed emerging memories for AI hardware accelerators. *Nature Reviews Electrical Engineering*, 1(1), 24-34, 2024.
    - A. Parashar, P. Raina; Y. S. Shao; Y.-H. Chen; V. A. Ying; A. Mukkara, Timeloop: A systematic approach to dnn accelerator evaluation. *2019 IEEE international symposium on performance analysis of systems and software (ISPASS)*. IEEE, 2019.

Step1: setting input parameters

Setting the input parameters under evaluate.py

- Input_file: the trace file under ./traces.
- tech_node: the technology node for TPU. Need to make sure that you have the performance parameters for the circuit modules in the parameters.py
- buffer_type: the buffer technology for the global buffer. SRAM, eDRAM, eNVMs are supported.
- buffer_size: the buffer size in terms of MB (default: 2 MB)
- buffer_bank_size: the bank size in MB (default: 256KB)
- buffer_subarray_size: the subarray size in MB (default: 256x512, each bank has 4x4 subarray)



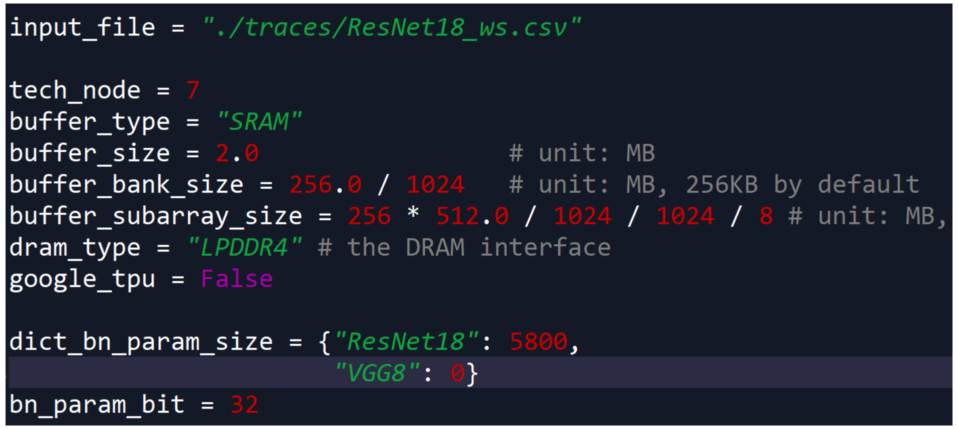
```
input_file = "./traces/ResNet18_ws.csv"

tech_node = 7
buffer_type = "SRAM"
buffer_size = 2.0                    # unit: MB
buffer_bank_size = 256.0 / 1024     # unit: MB, 256KB by default
buffer_subarray_size = 256 * 512.0 / 1024 / 1024 / 8 # unit: MB,
dram_type = "LPDDR4" # the DRAM interface
google_tpu = False

dict_bn_param_size = {"ResNet18": 5800,
                      "VGG8": 0}
bn_param_bit = 32
```

The user need to modify the parameters that are highlighted in red. Can just keep the other default parameters

Step1: setting input parameters

Setting the input parameters under evaluate.py

- dram_type: the dram interface. LPDDR3 or LPDDR4
- google_tpu: use similar design as Google's TPU v1.0 with 256x256 systolic array and 28MB on-chip SRAM buffer
- dict_bn_param_size: the number of bn parameters in each model. VGG8 is assumed to be trained with WAGE-mode without BN.
- bn_param_bit: the precision of bn parameters
- The size of the MAC array is not an input parameter as different MAC array size leads to different dataflow and thus different traces. Currently, 256x256 (Google's TPU) and 16x16 (edge) MAC arrays are included.

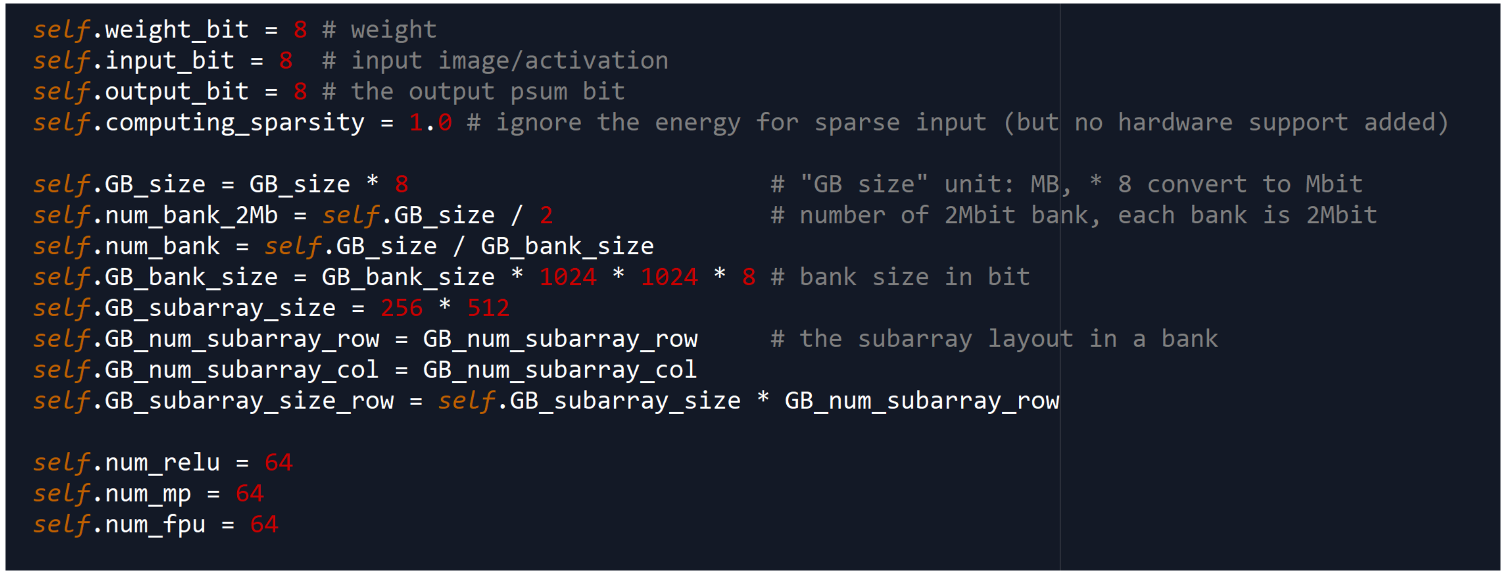


```
input_file = "./traces/ResNet18_ws.csv"

tech_node = 7
buffer_type = "SRAM"
buffer_size = 2.0                  # unit: MB
buffer_bank_size = 256.0 / 1024    # unit: MB, 256KB by default
buffer_subarray_size = 256 * 512.0 / 1024 / 1024 / 8 # unit: MB,
dram_type = "LPDDR4" # the DRAM interface
google_tpu = False

dict_bn_param_size = {"ResNet18": 5800,
                      "VGG8": 0}
bn_param_bit = 32
```

The user need to modify the parameters that are highlighted in red. Can keep the other default parameters

Step2: setting up DNN model parameters

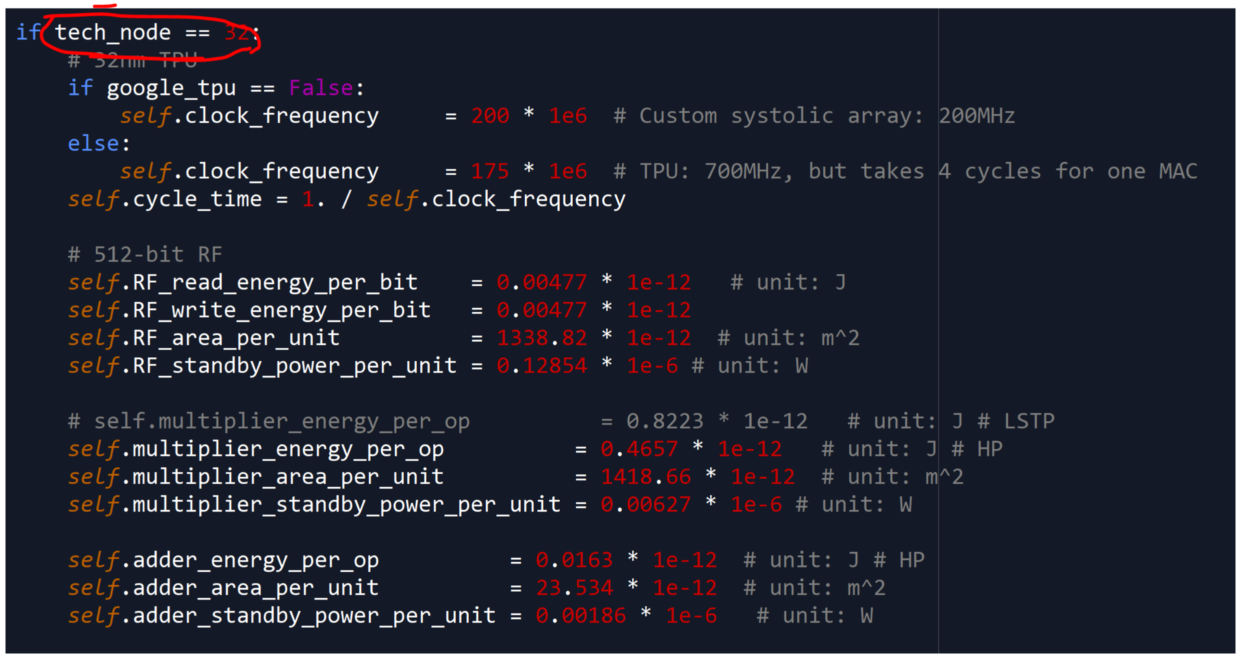
- Under parameters.py
- Set the precision of weight, input image, output activation, intermediate psums
- Set the computing sparsity. (but currently did not include the hardware overhead)
- Modify the number of relu, mp, fpus if you want



```
self.weight_bit = 8 # weight
self.input_bit = 8  # input image/activation
self.output_bit = 8 # the output psum bit
self.computing_sparsity = 1.0 # ignore the energy for sparse input (but no hardware support added)

self.GB_size = GB_size * 8                             # "GB size" unit: MB, * 8 convert to Mbit
self.num_bank_2Mb = self.GB_size / 2                   # number of 2Mbit bank, each bank is 2Mbit
self.num_bank = self.GB_size / GB_bank_size
self.GB_bank_size = GB_bank_size * 1024 * 1024 * 8 # bank size in bit
self.GB_subarray_size = 256 * 512
self.GB_num_subarray_row = GB_num_subarray_row     # the subarray layout in a bank
self.GB_num_subarray_col = GB_num_subarray_col
self.GB_subarray_size_row = self.GB_subarray_size * GB_num_subarray_row

self.num_relu = 64
self.num_mp = 64
self.num_fpu = 64
```

Step3: setting up circuit-module parameters

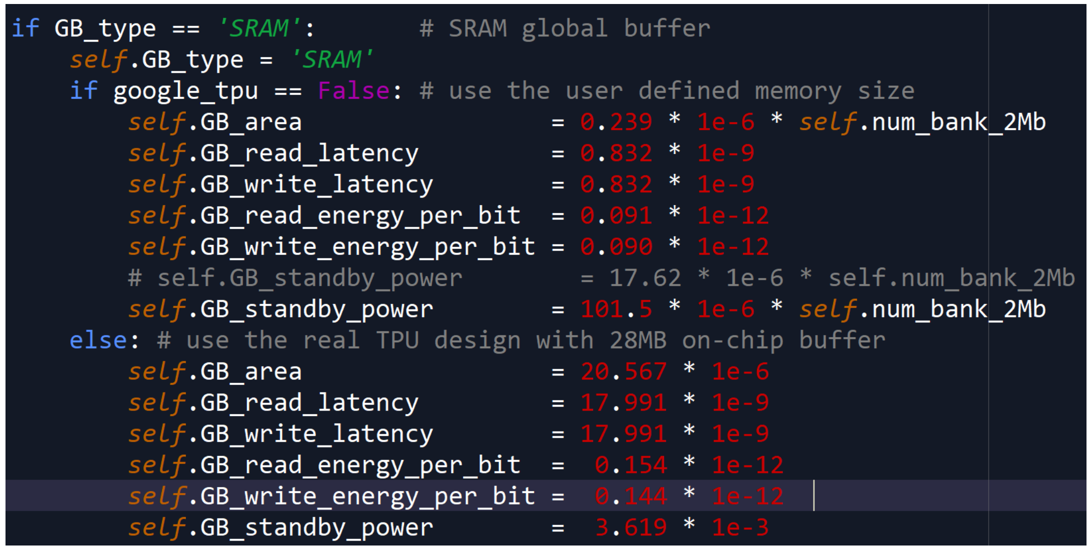
- Under parameters.py. Make sure that you are modifying the circuit module parameters **under the target technology node**.
- The area and stand-by power is for a 512-bit register file (RF)
- The area and stand-by power of multiplier and adder is for a single unit with 8-bit precision



```
if tech_node == 32:
    # 32nm TPU
    if google_tpu == False:
        self.clock_frequency     = 200 * 1e6  # Custom systolic array: 200MHz
    else:
        self.clock_frequency     = 175 * 1e6  # TPU: 700MHz, but takes 4 cycles for one MAC
    self.cycle_time = 1. / self.clock_frequency

    # 512-bit RF
    self.RF_read_energy_per_bit    = 0.00477 * 1e-12   # unit: J
    self.RF_write_energy_per_bit   = 0.00477 * 1e-12
    self.RF_area_per_unit          = 1338.82 * 1e-12  # unit: m^2
    self.RF_standby_power_per_unit = 0.12854 * 1e-6 # unit: W

    # self.multiplier_energy_per_op          = 0.8223 * 1e-12   # unit: J # LSTP
    self.multiplier_energy_per_op          = 0.4657 * 1e-12   # unit: J # HP
    self.multiplier_area_per_unit          = 1418.66 * 1e-12  # unit: m^2
    self.multiplier_standby_power_per_unit = 0.00627 * 1e-6 # unit: W

    self.adder_energy_per_op          = 0.0163 * 1e-12  # unit: J # HP
    self.adder_area_per_unit          = 23.534 * 1e-12  # unit: m^2
    self.adder_standby_power_per_unit = 0.00186 * 1e-6   # unit: W
```

Step4: setting up global buffer parameters

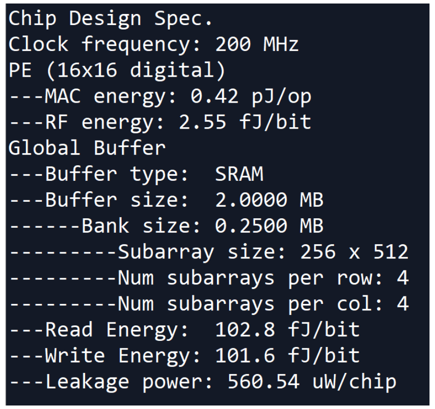
- The global buffer is assumed to consist of multiple 256KB (2Mb) banks. Therefore, the area, latency, energy/bit, standby-power are that for a **256KB bank**.
  - It should be noted that the interconnect energy is included for both read and write
- You can obtain the performance of a 256KB bank using simulators for on-chip memory or buffer (e.g. NVSim, Destiny) or using NeuroSim (but only SRAM buffer is available).



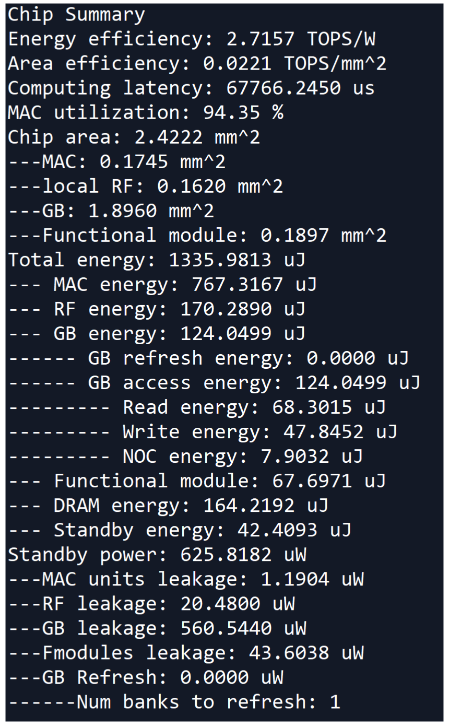
```
if GB_type == 'SRAM':        # SRAM global buffer
    self.GB_type = 'SRAM'
    if google_tpu == False: # use the user defined memory size
        self.GB_area                = 0.239 * 1e-6 * self.num_bank_2Mb
        self.GB_read_latency        = 0.832 * 1e-9
        self.GB_write_latency       = 0.832 * 1e-9
        self.GB_read_energy_per_bit  = 0.091 * 1e-12
        self.GB_write_energy_per_bit = 0.090 * 1e-12
        # self.GB_standby_power        = 17.62 * 1e-6 * self.num_bank_2Mb
        self.GB_standby_power       = 101.5 * 1e-6 * self.num_bank_2Mb
    else: # use the real TPU design with 28MB on-chip buffer
        self.GB_area                = 20.567 * 1e-6
        self.GB_read_latency        = 17.991 * 1e-9
        self.GB_write_latency       = 17.991 * 1e-9
        self.GB_read_energy_per_bit  =  0.154 * 1e-12
        self.GB_write_energy_per_bit =  0.144 * 1e-12
        self.GB_standby_power       =  3.619 * 1e-3
```

Step5: running and getting the stats

- Python evaluate.py
- The stats will be printed automatically. Below is an exemplar output
- The code has been tested with python 3 and numpy 1.13



```
Chip Design Spec.
Clock frequency: 200 MHz
PE (16x16 digital)
---MAC energy: 0.42 pJ/op
---RF energy: 2.55 fJ/bit
Global Buffer
---Buffer type:  SRAM
---Buffer size:  2.0000 MB
------Bank size: 0.2500 MB
---------Subarray size: 256 x 512
---------Num subarrays per row: 4
---------Num subarrays per col: 4
---Read Energy:  102.8 fJ/bit
---Write Energy: 101.6 fJ/bit
---Leakage power: 560.54 uW/chip
```



```
Chip Summary
Energy efficiency: 2.7157 TOPS/W
Area efficiency: 0.0221 TOPS/mm^2
Computing latency: 67766.2450 us
MAC utilization: 94.35 %
Chip area: 2.4222 mm^2
---MAC: 0.1745 mm^2
---local RF: 0.1620 mm^2
---GB: 1.8960 mm^2
---Functional module: 0.1897 mm^2
Total energy: 1335.9813 uJ
--- MAC energy: 767.3167 uJ
--- RF energy: 170.2890 uJ
--- GB energy: 124.0499 uJ
------ GB refresh energy: 0.0000 uJ
------ GB access energy: 124.0499 uJ
--------- Read energy: 68.3015 uJ
--------- Write energy: 47.8452 uJ
--------- NOC energy: 7.9032 uJ
--- Functional module: 67.6971 uJ
--- DRAM energy: 164.2192 uJ
--- Standby energy: 42.4093 uJ
Standby power: 625.8182 uW
---MAC units leakage: 1.1904 uW
---RF leakage: 20.4800 uW
---GB leakage: 560.5440 uW
---Fmodules leakage: 43.6038 uW
---GB Refresh: 0.0000 uW
------Num banks to refresh: 1
```