http://synergy.ece.gatech.edu

Tutorial 3: Scale-Sim as a library

ASPLOS-2021

Apr 16, 2021

Objective

To demonstrate Scale-sim as a library to build other simulators

Step 0: Check python version

- In your virtual environment please ensure python3

```
(scale)
2.Tutorials/ASPLOS-Tutorial/tutorial3
▶ python -V
Python 3.9.4
```

- Install scalesim package

```
> pip install scalesim
```

Overview

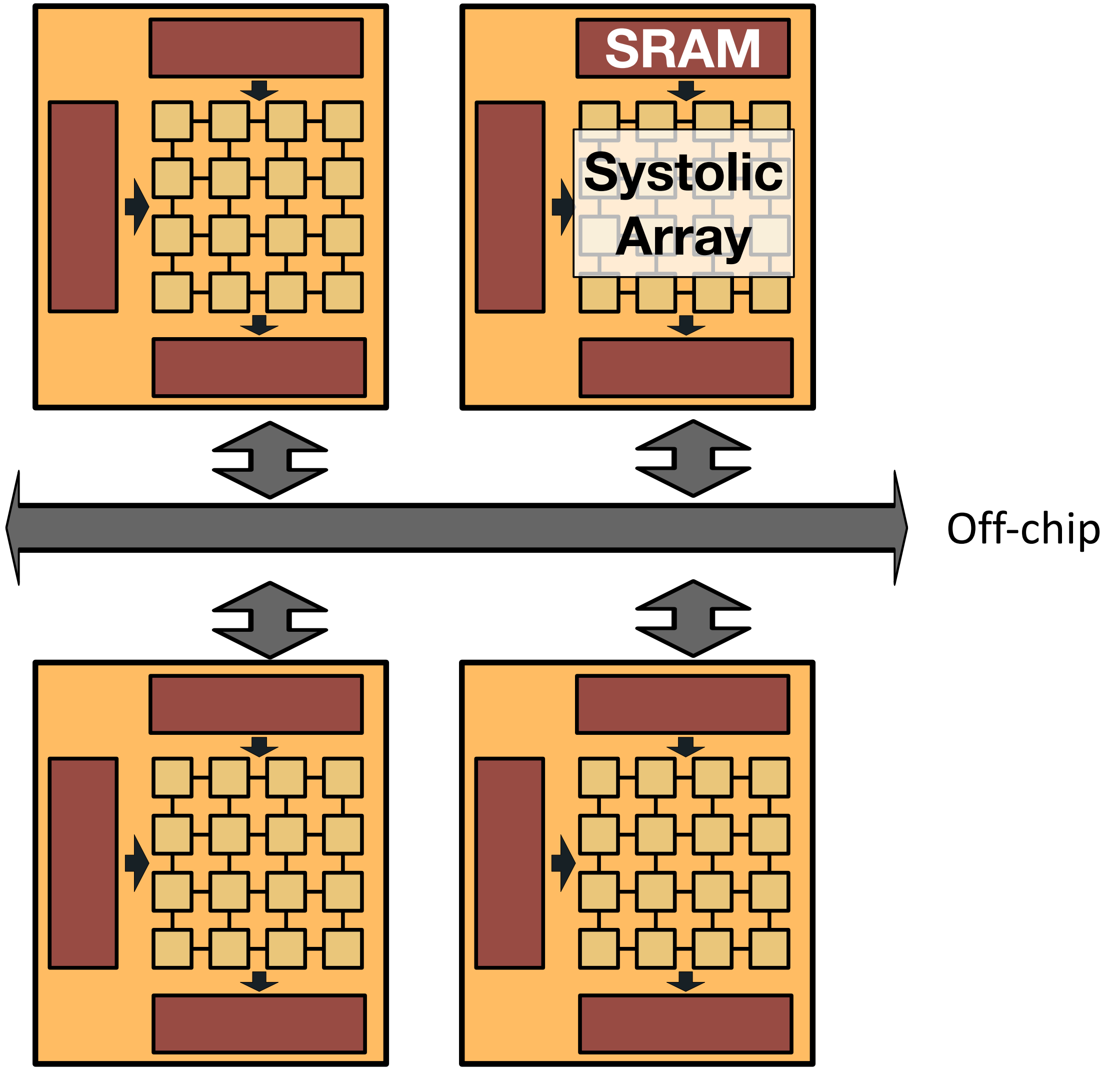
1. **High level description of the simulator.**

2. Building the simulator.

3. Simulation runs

4. Discussion

High level description



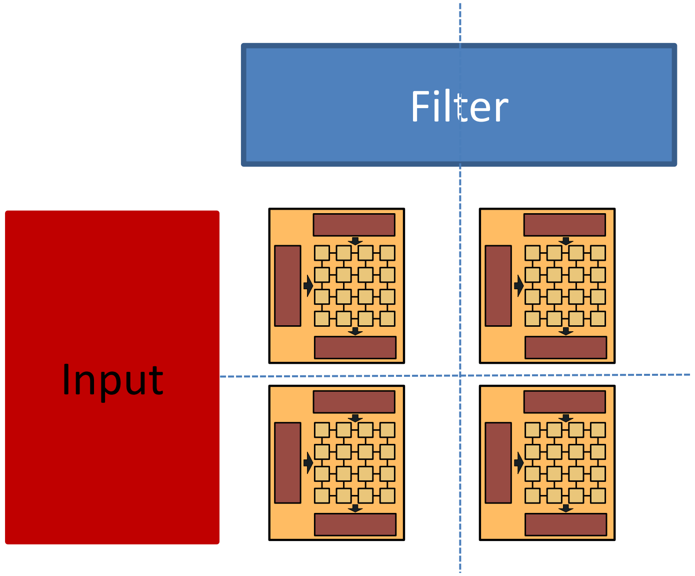
System: A four unit distributed array

Array dimensions are same as tutorial 1

Parameter	Value
Rows	64
Cols	64
IFMAP SRAM Size	512 KB
Filter SRAM Size	512 KB
OFMAP SRAM Size	512 KB
Dataflow	Output stationary

High level description (cont'd)

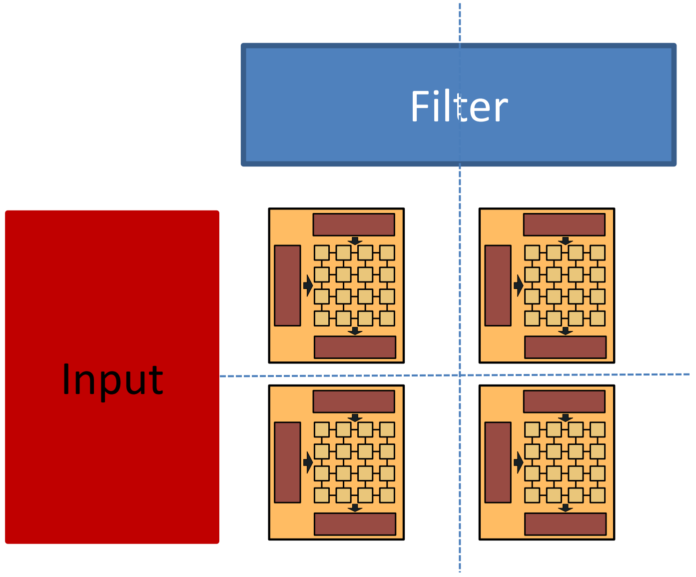
We will examine 3 logical layouts



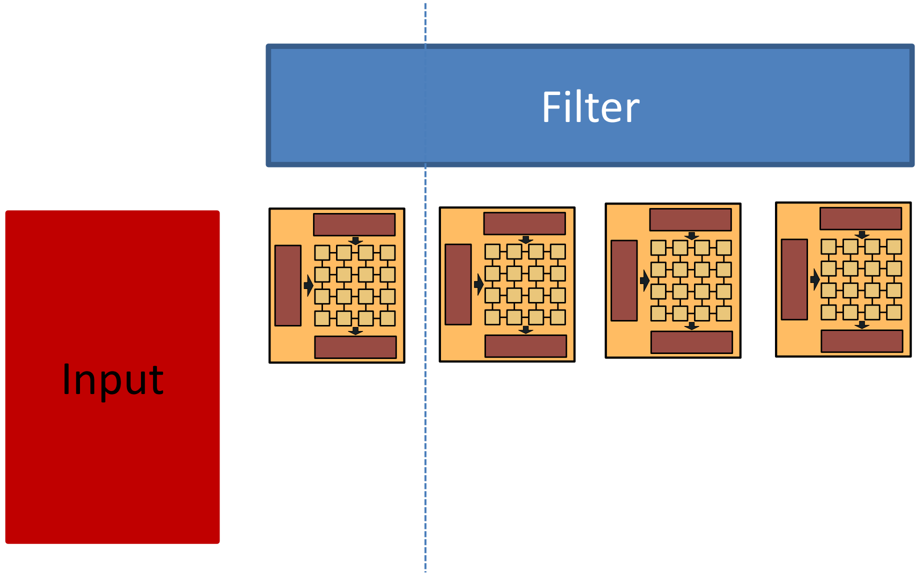
2x2 grid

High level description (cont'd)

We will examine 3 logical layouts



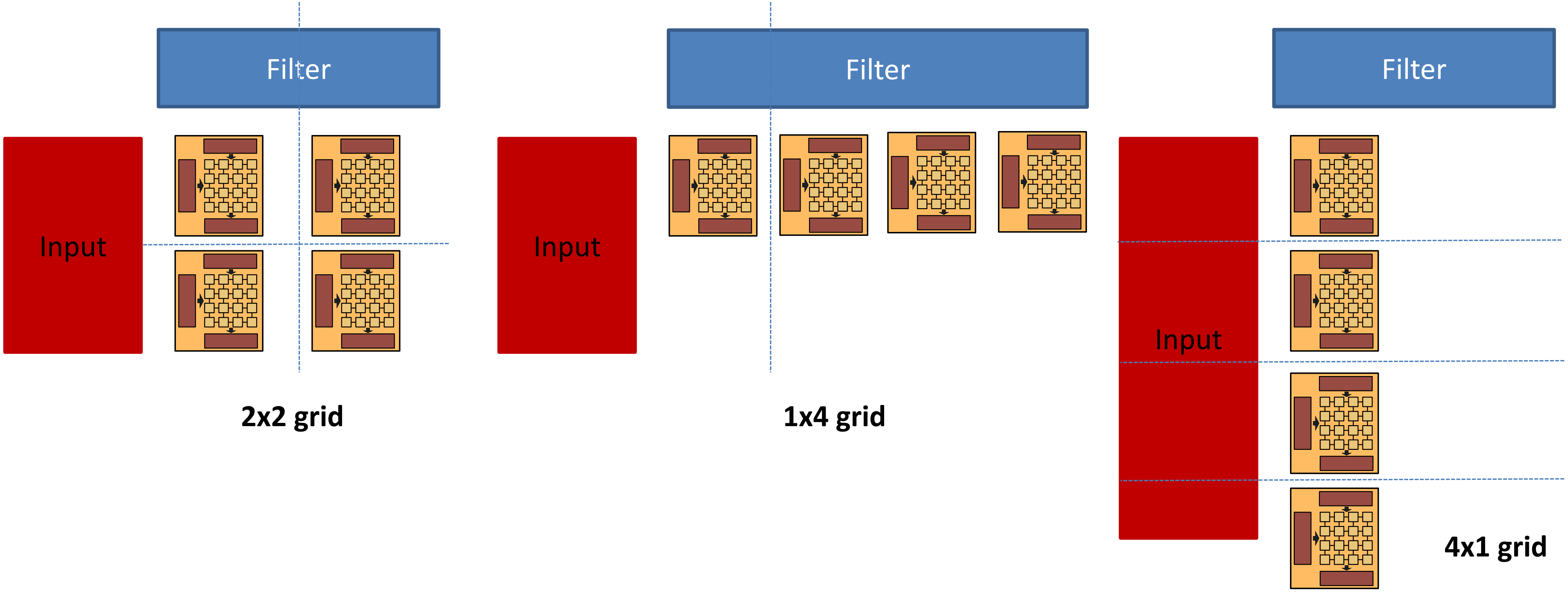
2x2 grid



1x4 grid

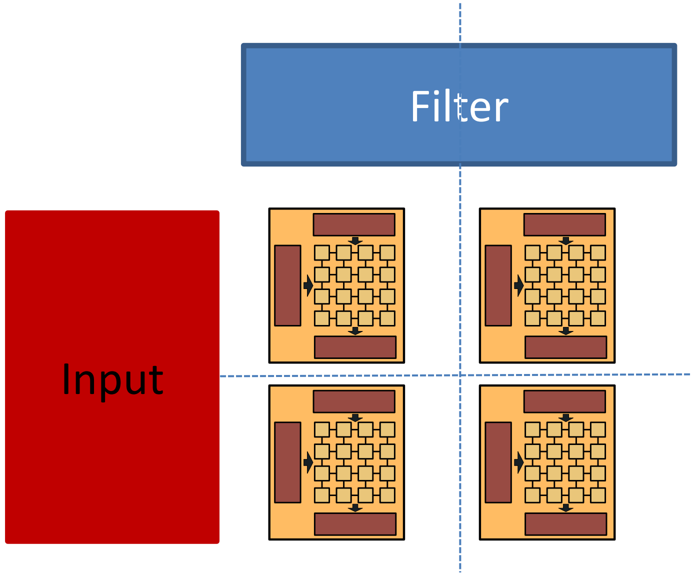
High level description (cont'd)

We will examine 3 logical layouts

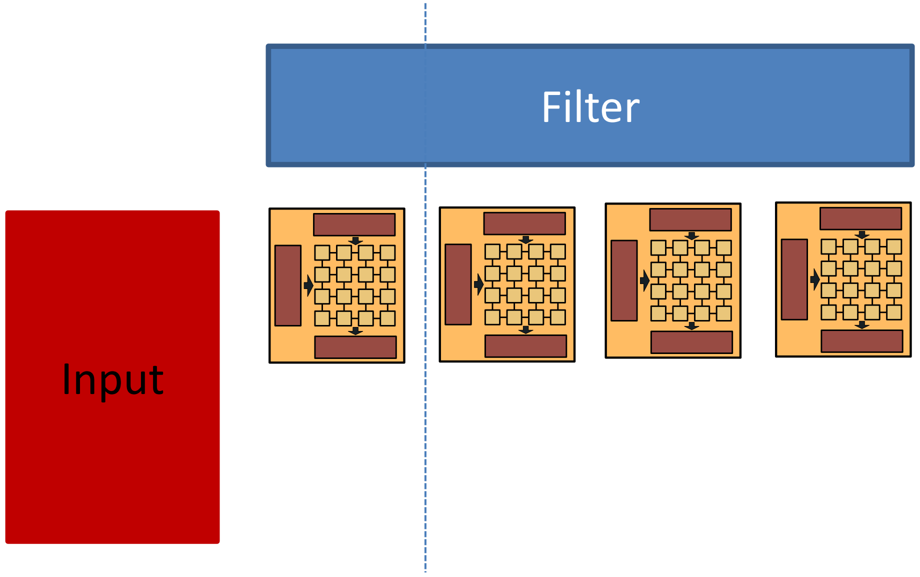


High level description (cont'd)

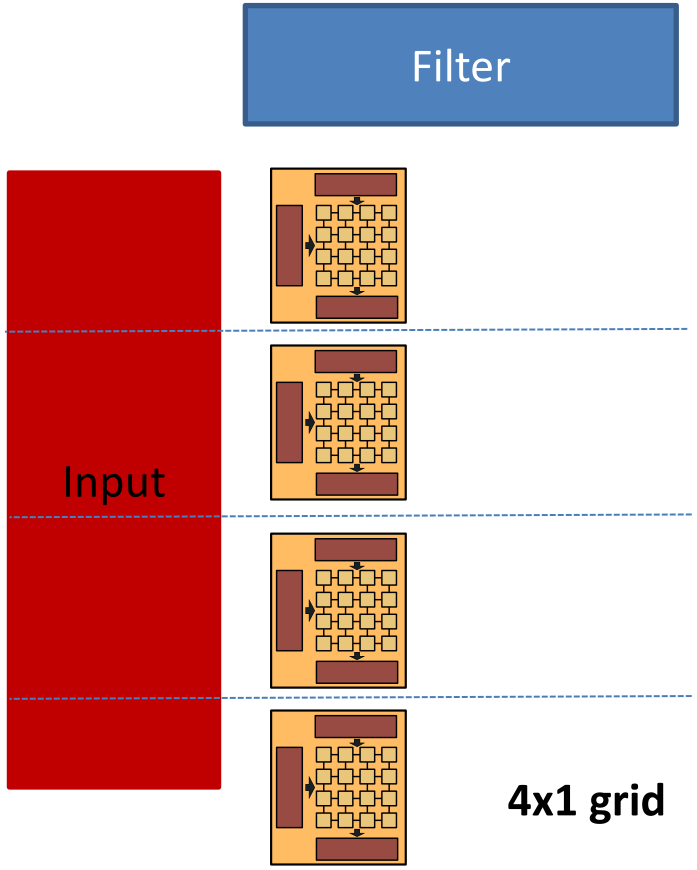
We will examine 3 logical layouts



2x2 grid



1x4 grid



4x1 grid

Metrics of interest:

1. Performance
2. Mapping
3. Off chip access

Overview

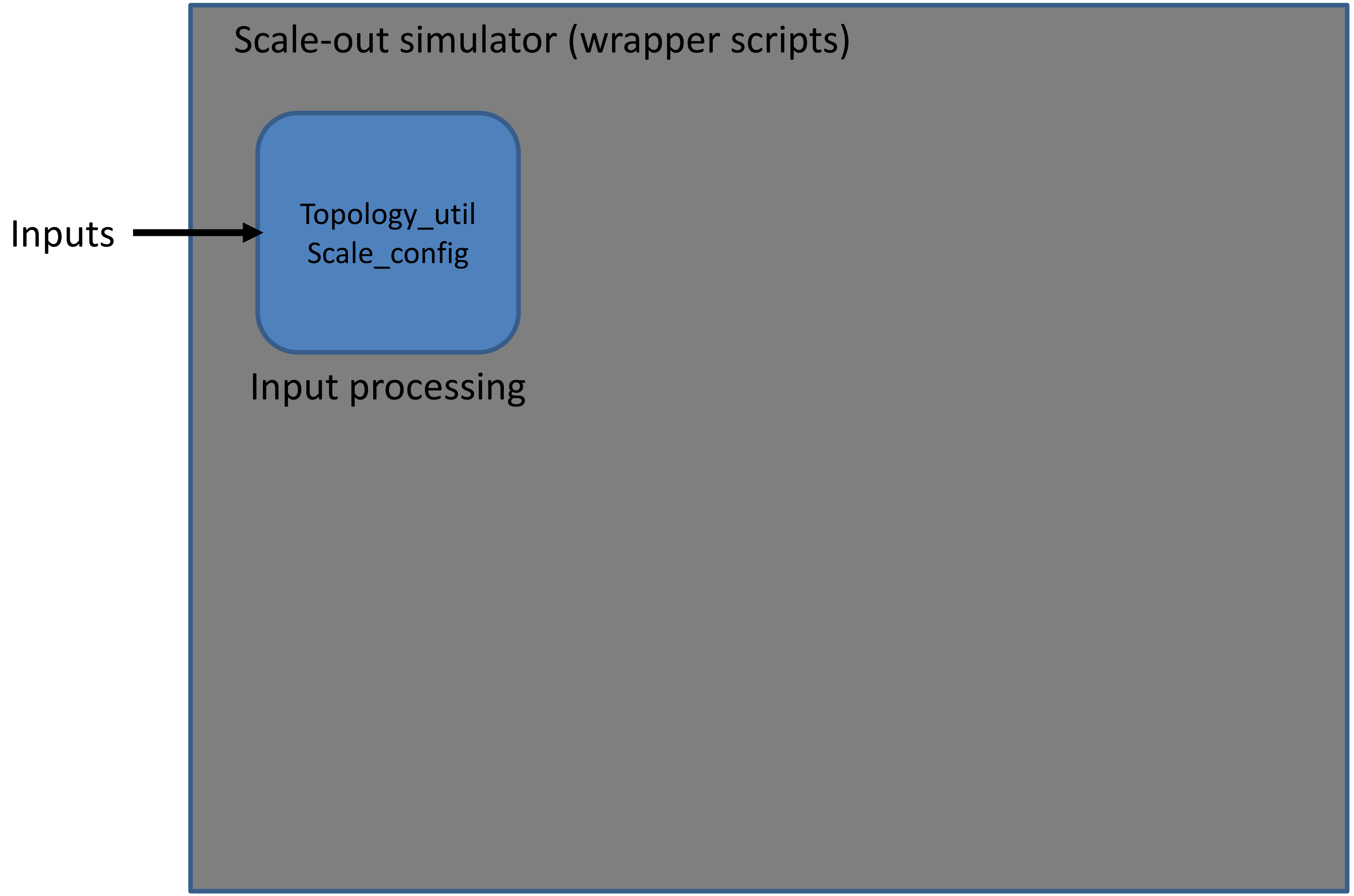
1. High level description of the simulator.

2. **Building the simulator**.

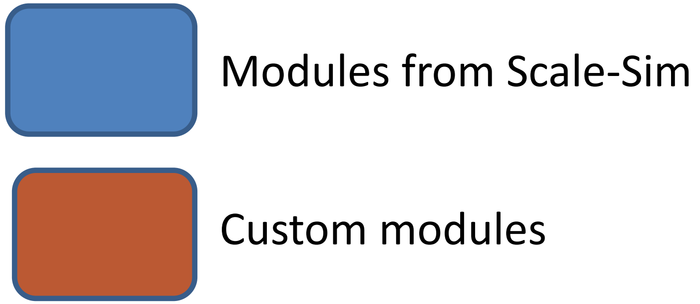
3. Simulation runs

4. Discussion

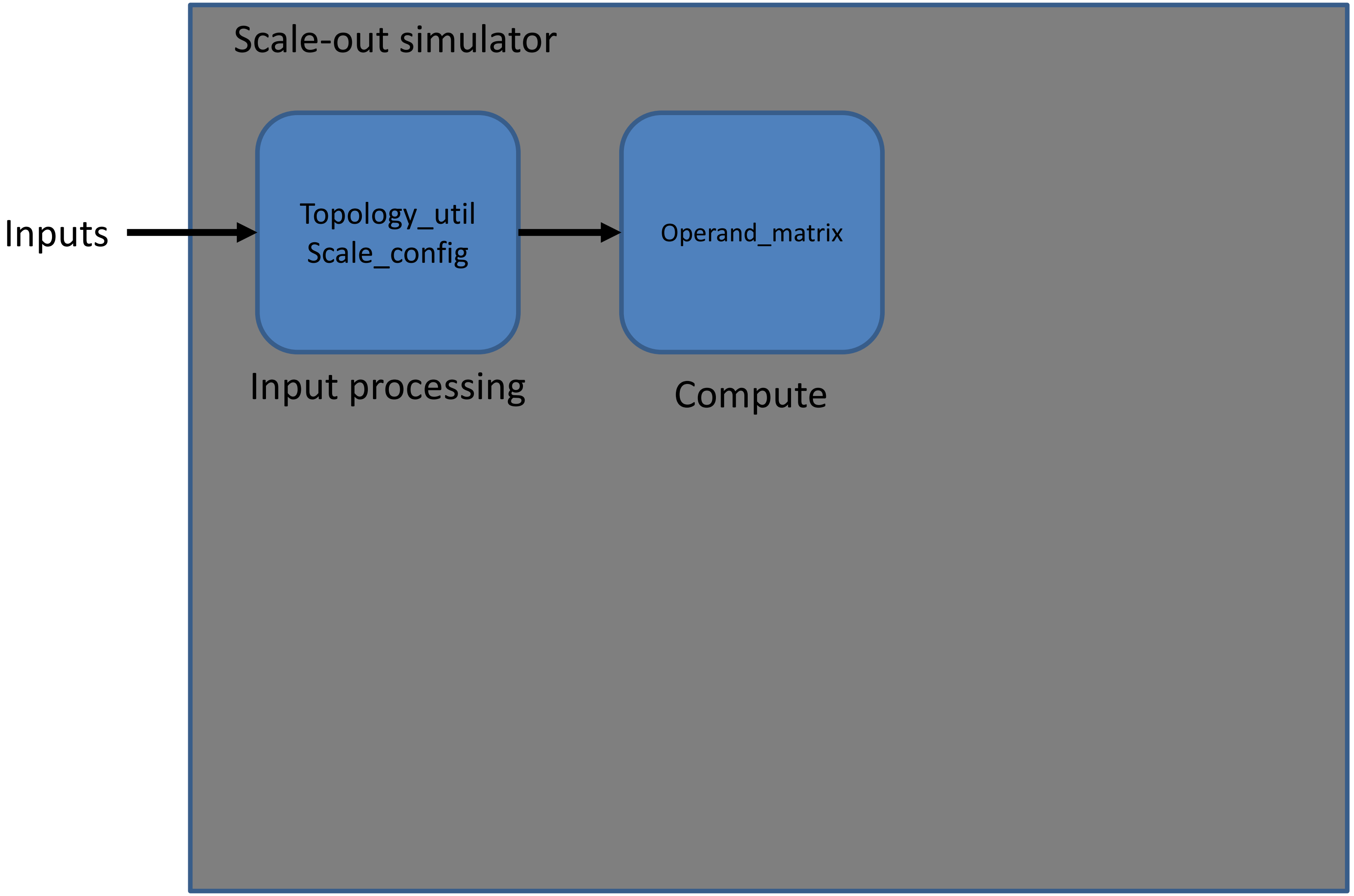
Implementation plan



Objective: Maximize code reuse



Implementation plan

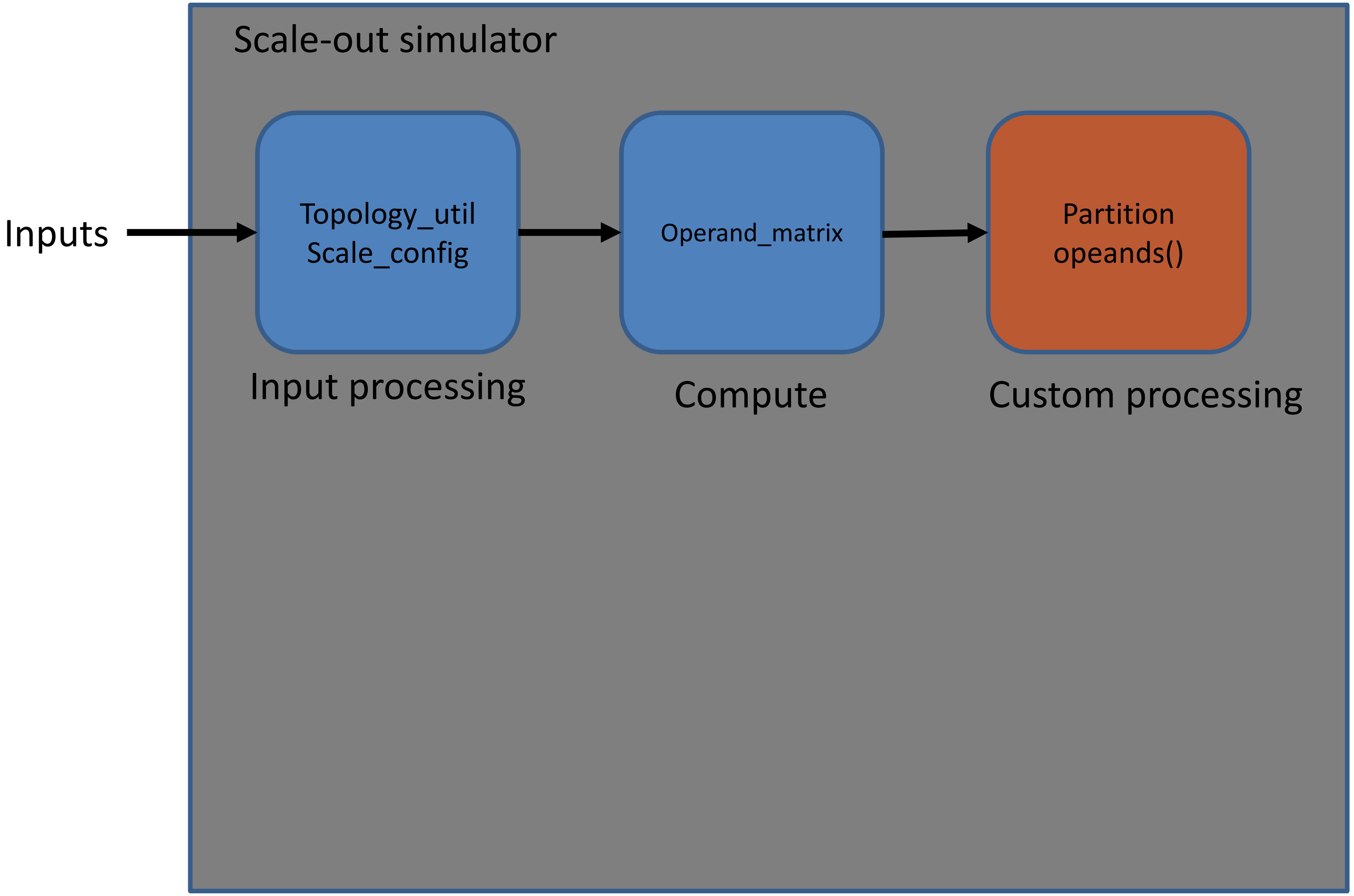


Objective: Maximize code reuse

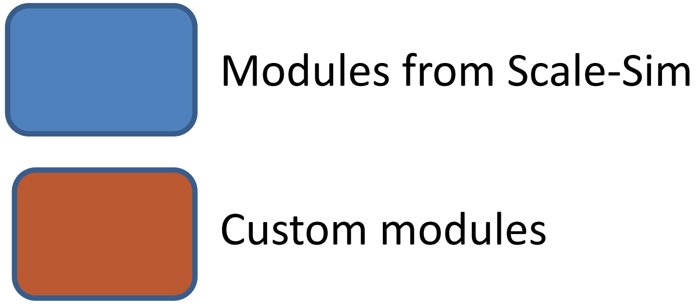
Modules from Scale-Sim

Custom modules

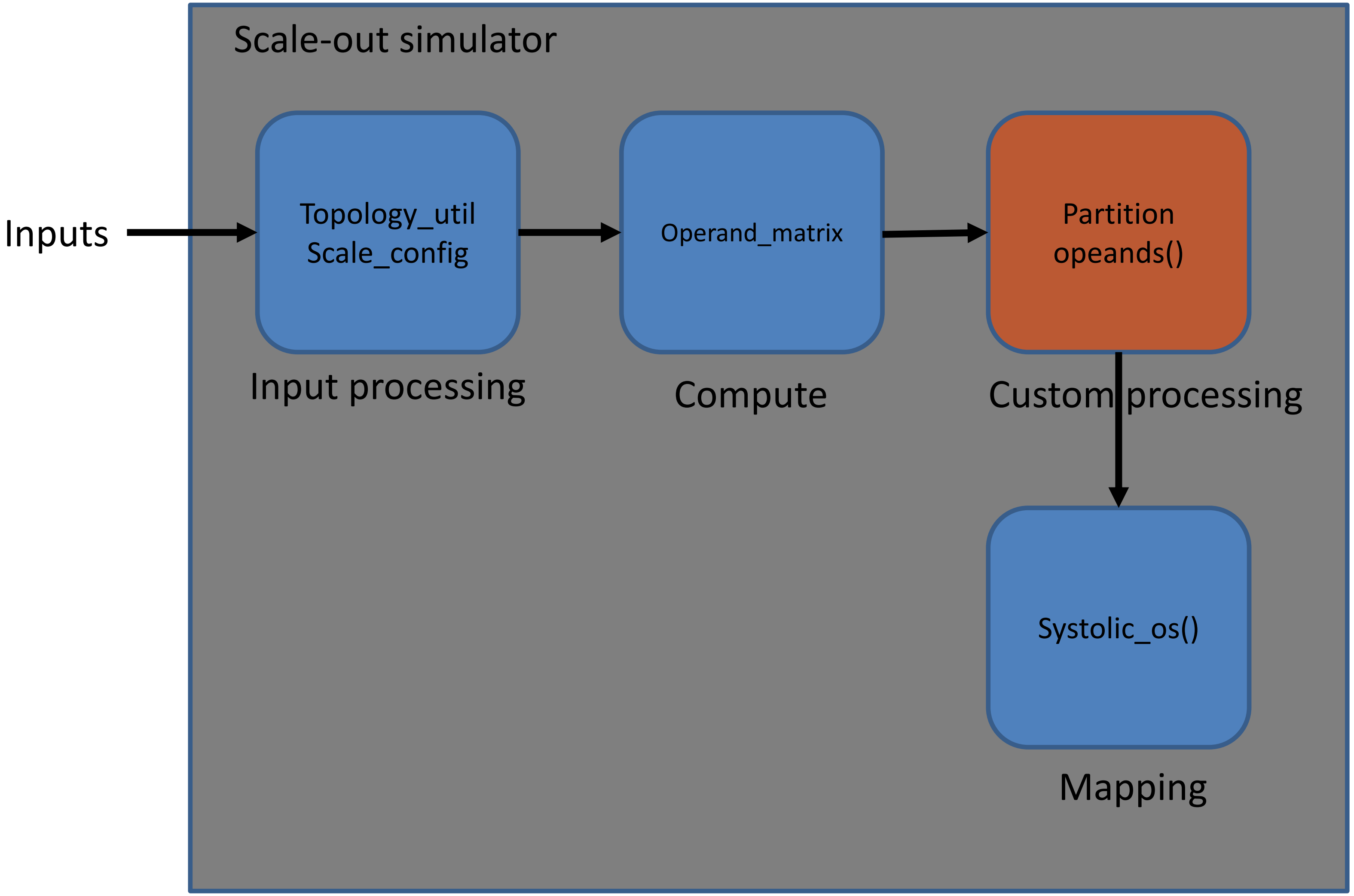
Implementation plan



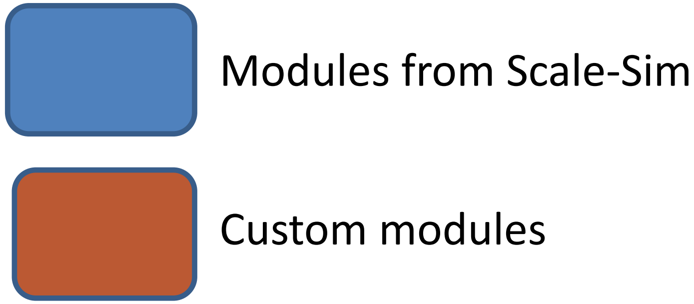
Objective: Maximize code reuse



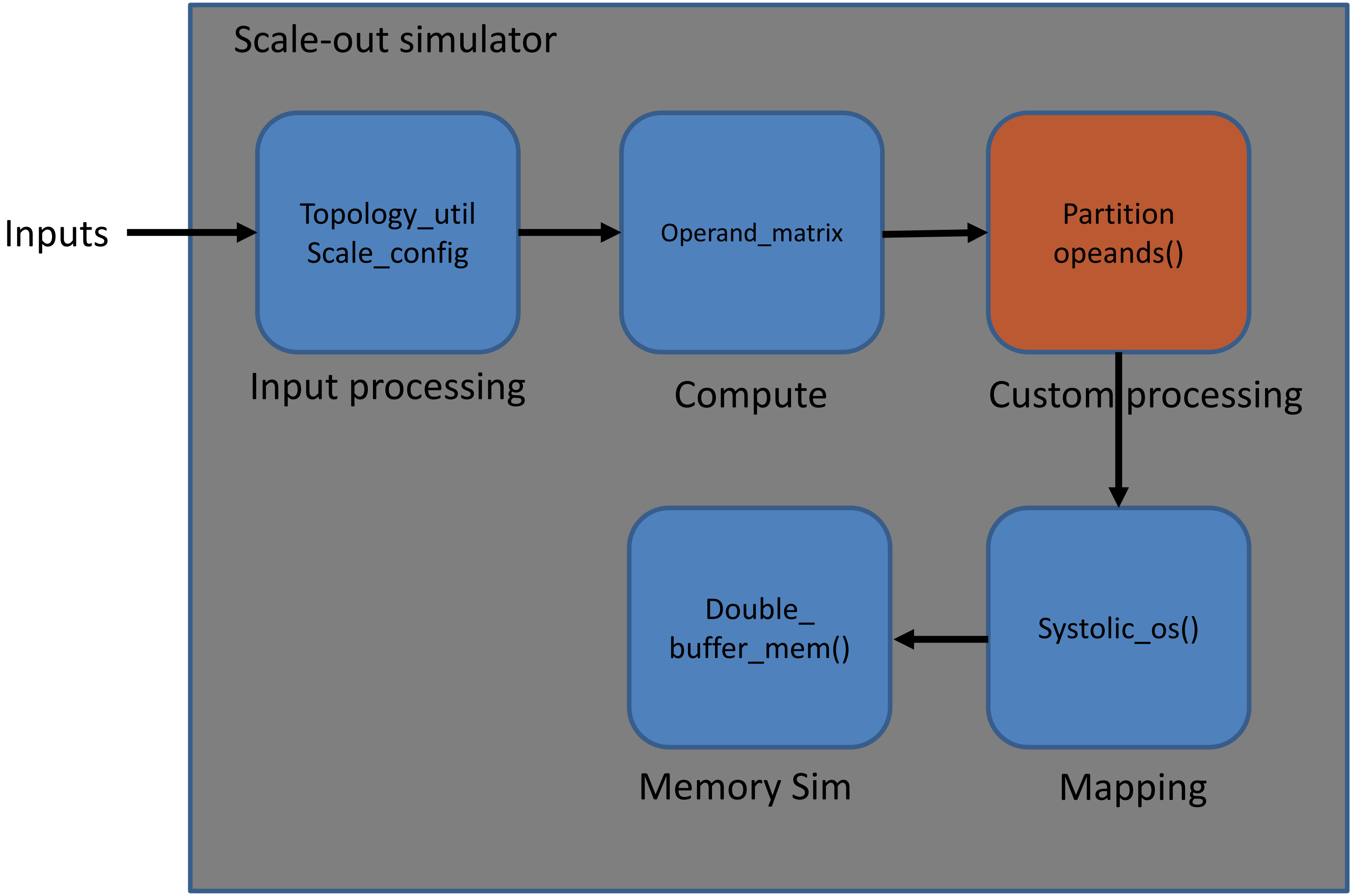
Implementation plan



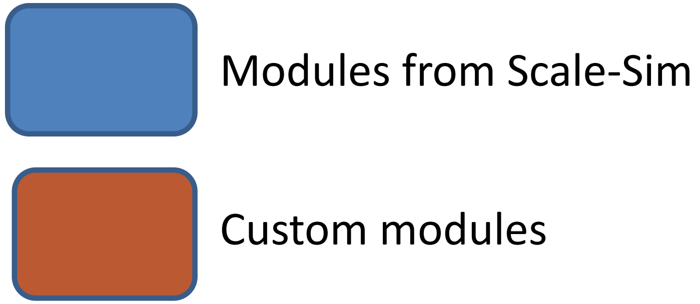
Objective: Maximize code reuse



Implementation plan

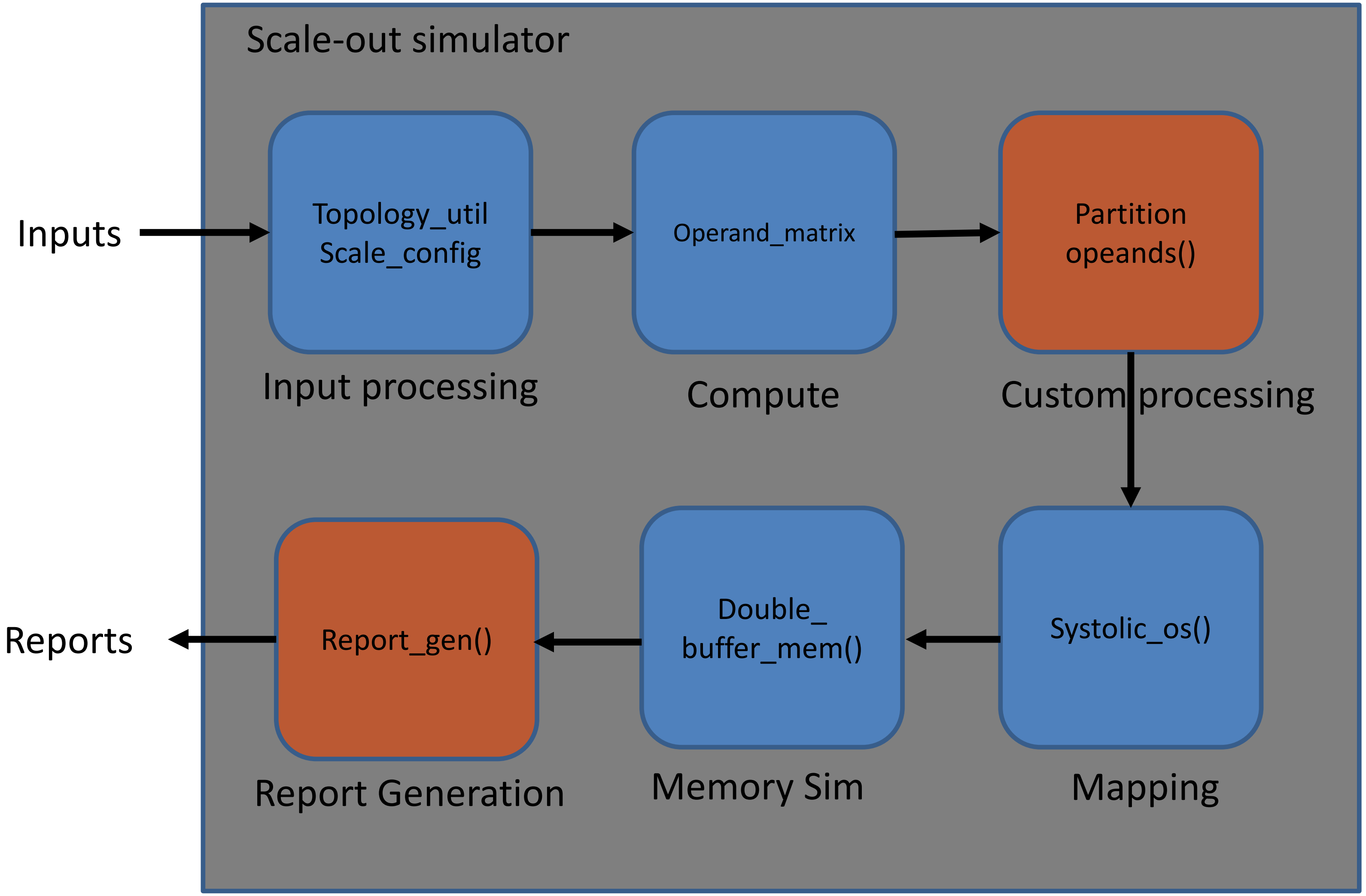


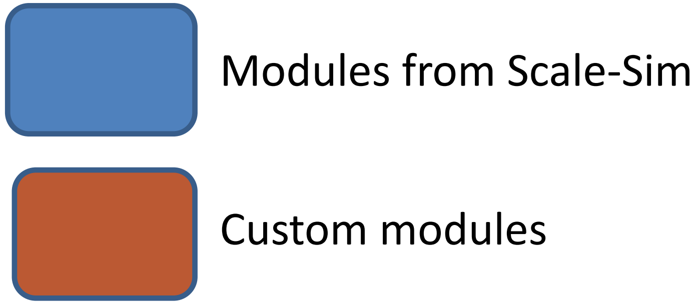
Objective: Maximize code reuse



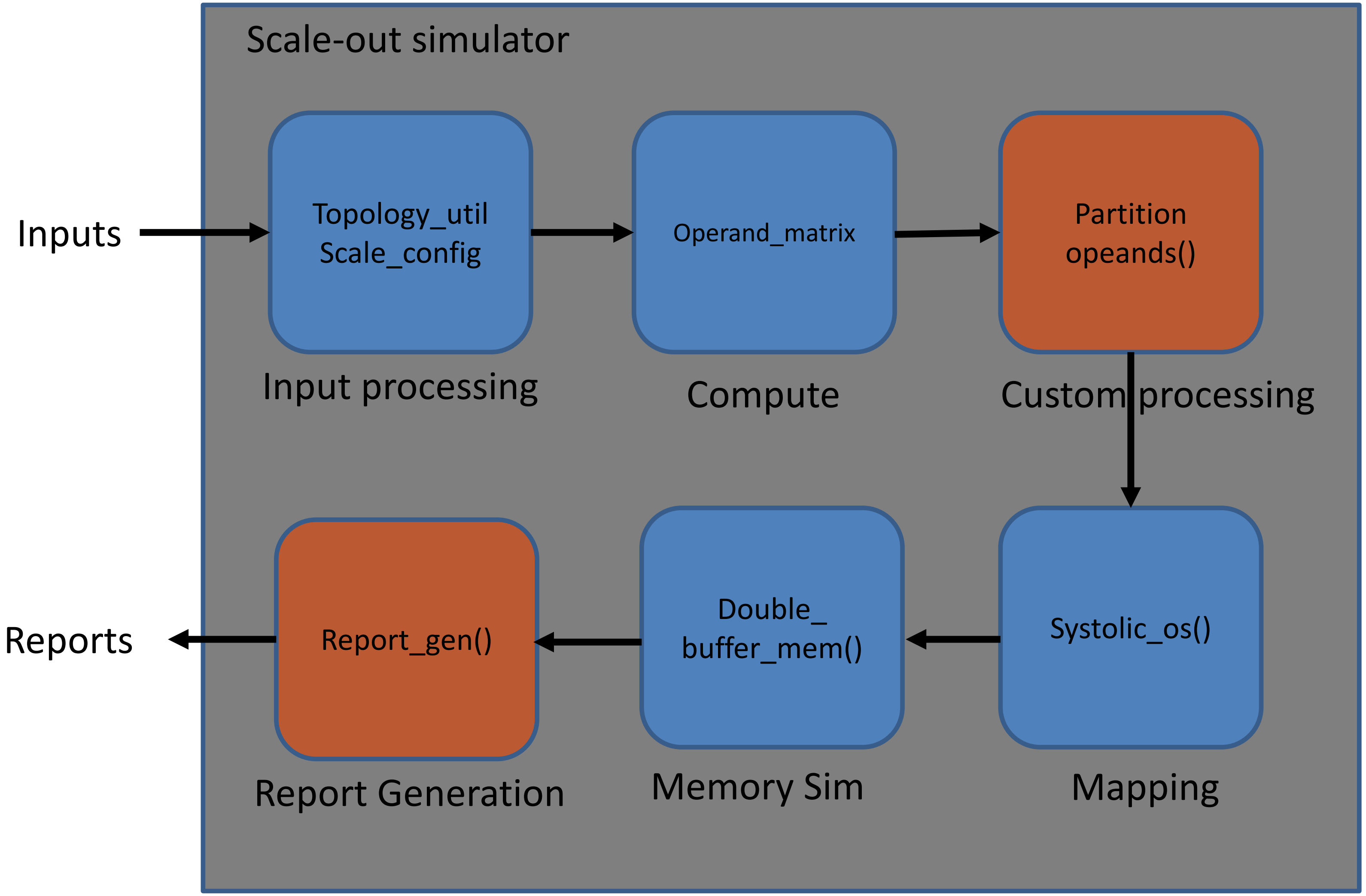
Implementation plan

Objective: Maximize code reuse





Implementation plan



Objective: Maximize code reuse

70% module reuse

Modules from Scale-Sim

Custom modules

Code

Step 1: Invoke utilities to read the inputs

Add your code for Blank 1 at Line 62

```
#
def set_params(self,
                topology_filename='./files/tutorial3_topofile.csv',
                single_arr_config_file='./files/single_arr_config.cfg',
                grid_rows=1, grid_cols=1,
                dataflow = 'os'
                ):

    # Blank 1. Read the input files
    # <insert code here>
```

Code to add

```
## Code for Blank 1

self.topo_obj = topologies()
self.topo_obj.load_arrays(topology_filename)
num_layers=self.topo_obj.get_num_layers()

self.single_arr_cfg = scale_config()
self.single_arr_cfg.read_conf_file(single_arr_config_file)
```

Code

Step 2: Invoke utilities to create operand matrices

Add your code for Blank 2 at Line 99

```
4      #
3      def run_simulation_single_layer(self, layer_id=0):
2
1          # Blank 2. Create the operand matrices
99         # <Insert code here>
```

Code to add

```
## Code for Blank 2

opmat_obj = opmat()
opmat_obj.set_params(config_obj=self.single_arr_cfg, topoutil_obj=self.topo_obj, layer_id=layer_id)

_, ifmap_op_mat = opmat_obj.get_ifmap_matrix()
_, filter_op_mat = opmat_obj.get_filter_matrix()
_, ofmap_op_mat = opmat_obj.get_ofmap_matrix()

```

Code

Step 3: Instantiate the compute unit

Add your code for Blank 3 at Line 112

```
8               arr_id = grid_row_id * self.grid_cols + grid_col_id
7               print('Running subarray ' + str(arr_id))
6
5               ifmap_op_mat_part, filter_op_mat_part, ofmap_op_mat_part =\
4                   self.get_opmat_parts(ifmap_op_mat, filter_op_mat, ofmap_op_mat,
3                                        grid_row_id, grid_col_id)
2
1               # Blank 3. Instantiate the mapping utilities
112             #<Insert code here>
```

Code to add

```
##Code for Blank 3

compute_system = systolic_compute_os()
if self.dataflow == 'ws':
    compute_system = systolic_compute_ws()
elif self.dataflow == 'is':
    compute_system = systolic_compute_is()

compute_system.set_params(config_obj=self.single_arr_cfg,
                          ifmap_op_mat=ifmap_op_mat_part,
                          filter_op_mat=filter_op_mat_part,
                          ofmap_op_mat=ofmap_op_mat_part)

ifmap_demand_mat, filter_demand_mat, ofmap_demand_mat = compute_system.get_demand_matrices()
```

Code

Step 4: Instantiate the memory

Add your code for Blank 4 at Line 112

```
  1
114                 # Blank 4. Memory system
  1                 #<Insert code here>
  2
  3                 self.gather_stats(row_id=grid_row_id,
  4                                   col_id=grid_col_id,
  5                                   memory_system_obj=memory_system,
  6                                   layer_id=layer_id)
  7
  8         self.all_grids_done = True
  9
 10     #
 11     def run_simulations_all_layers(self):
 12         assert self.params_valid, 'Params are not valid'
 13
```

Code

Step 4: Instantiate the memory

Code to add

```
##Code for Blank 4

memory_system = mem_dbsp()

ifmap_buf_size_kb, filter_buf_size_kb, ofmap_buf_size_kb = self.single_arr_cfg.get_mem_sizes()
ifmap_buf_size_bytes = 1024 * ifmap_buf_size_kb
filter_buf_size_bytes = 1024 * filter_buf_size_kb
ofmap_buf_size_bytes = 1024 * ofmap_buf_size_kb

arr_row, arr_col = self.single_arr_cfg.get_array_dims()

ifmap_backing_bw = 1
filter_backing_bw = 1
ofmap_backing_bw = 1
if self.dataflow == 'os' or self.dataflow == 'ws':
    ifmap_backing_bw = arr_row
    filter_backing_bw = arr_col
    ofmap_backing_bw = arr_col

elif self.dataflow == 'is':
    ifmap_backing_bw = arr_col
    filter_backing_bw = arr_row
    ofmap_backing_bw = arr_col

memory_system.set_params(
    word_size=1,
    ifmap_buf_size_bytes=ifmap_buf_size_bytes,
    filter_buf_size_bytes=filter_buf_size_bytes,
    ofmap_buf_size_bytes=ofmap_buf_size_bytes,
    rd_buf_active_frac=0.5, wr_buf_active_frac=0.5,
    ifmap_backing_buf_bw=ifmap_backing_bw,
    filter_backing_buf_bw=filter_backing_bw,
    ofmap_backing_buf_bw=ofmap_backing_bw,
    verbose=True,
    estimate_bandwidth_mode=True
)

memory_system.service_memory_requests(ifmap_demand_mat, filter_demand_mat, ofmap_demand_mat)
```

Overview

1. High level description of the simulator.
2. Building the simulator.
3. **Simulation runs**
4. Discussion

Command

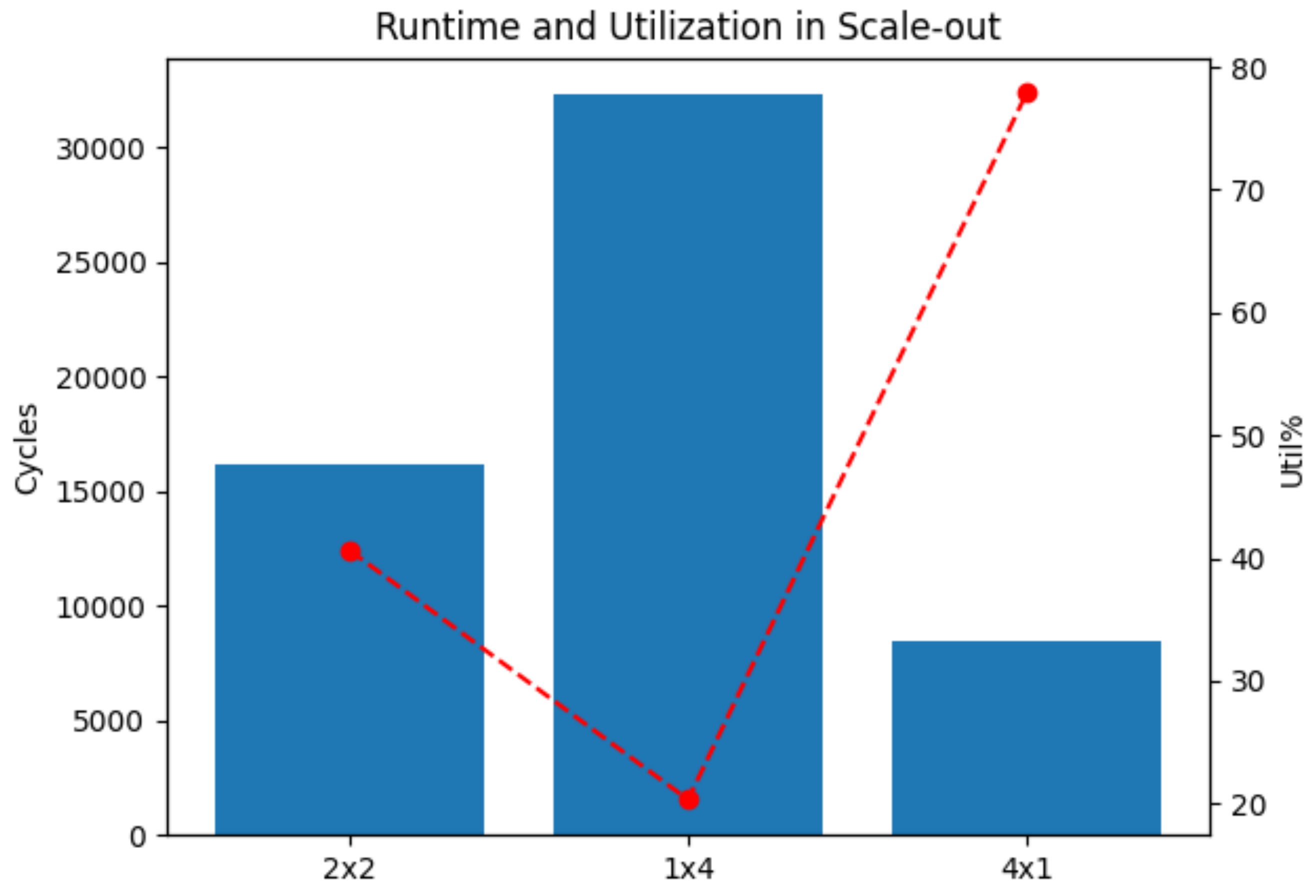
Run the completed simulator using the following command

```
> python tutorial3.py
```

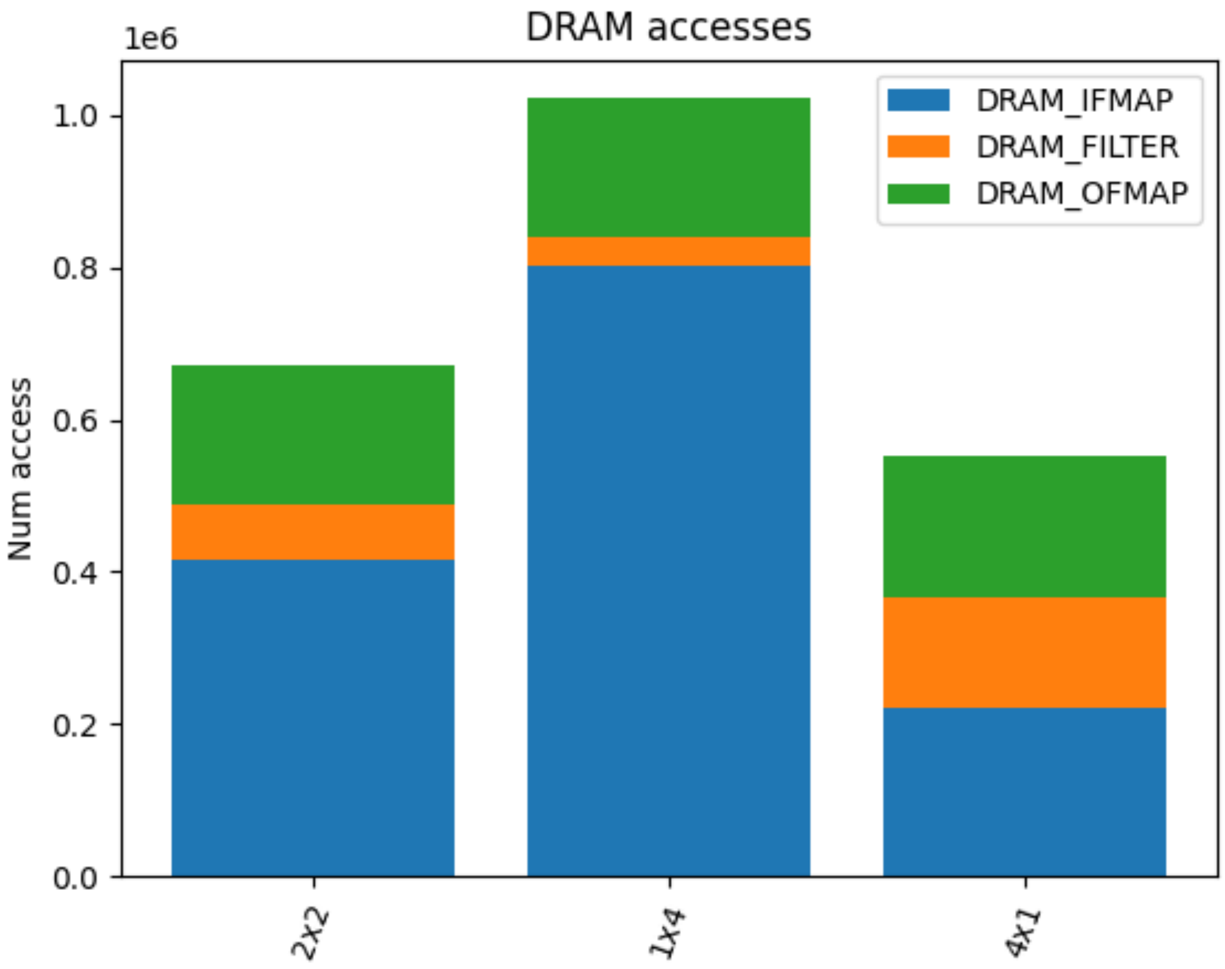
Overview

1. High level description of the simulator.
2. Building the simulator.
3. Simulation runs
4. **Discussion**

Analysis of runtime and Utilization



Analysis of DRAM accesses



Observations

1. 4x1 is better in terms of DRAM accesses as well
2. The difference in DRAM accesses for same operand suggest that there is better reuse in 4x1

SCALE-Sim library support accelerates building custom simulators