http://synergy.ece.gatech.edu

Tutorial 1: SCALE-Sim as a package

ASPLOS-2021

Apr 16, 2021

Setting up

Please download the resource files from the tutorial webpage

https://scalesim-project.github.io/tutorials-2021-asplos.html

OR

https://git.io/JOCFC

metrics as outputs

Schedule (April 16, 2021; Eastern Time)

Time	Agenda	Presenter
10:10-10:45	Introduction to DNNs and Accelerator Design	Tushar, Paul
10:45-11:15	Overview of SCALE-Sim	Paul, Anand, Moritz
11:15-11:50	Tutorial 1: Design Space Exploration using SCALE-Sim	Anand
noon-12:40	Tutorial 2: Modifying SCALE-Sim to add custom features	Moritz
12:45-1:30	Tutorial 3: Using SCALE-Sim to build larger simulators	Anand
1:30-2:00	Plenum: Discussion on future roadmap, planned features, and a ideas from the community	Yuhao

Resources

- Files for tutorial
- SCALE-Sim v2 Github
- SCALE-Sim pip repo

Link

Demo 0: Scale-Sim first run

We will show how to quickly get started with SCALE-Sim

SCALE-Sim can be setup in three easy steps:

1. Setup using pip install
2. Preparing the inputs
3. Launching the simulations

Step 0: Check python version

- If you are working on your machine

Step 1: Setup

Download and install the tool using pip

```
pip install scalesim
```

```
[ ]  !pip install scalesim
```

```
Collecting scalesim
  Downloading https://files.pythonhosted.org/packages/68/cf/c8f6f230b830f02f18707d4a94d48bd4d05a94d055309abdd9452e7
     |████████████████████████████████| 51kB 6.2MB/s
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from scalesim) (1.19.5)
Collecting configparser
  Downloading https://files.pythonhosted.org/packages/fd/01/ff260a18caaf4457eb028c96eeb405c4a230ca06c8ec9c1379f813c
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (from scalesim) (4.41.1)
Requirement already satisfied: absl-py in /usr/local/lib/python3.7/dist-packages (from scalesim) (0.12.0)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from absl-py->scalesim) (1.15.0)
Installing collected packages: configparser, scalesim
Successfully installed configparser-5.0.2 scalesim-2.0
```

Step2: Preparing the inputs

```python
from scalesim.scale_sim import scalesim

content_path = "/content/drive/MyDrive/scalesim_resources"
config = content_path + "/configs/scale.cfg"
topo = content_path + "/topologies/conv_nets/alexnet_part.csv"
```

Import the package to your script

Path to the config file

- Array Dimensions
- Dataflow
- Memory Sizes
- Run mode

Path to the topology csv file

- Layer wise workload parameters

```python
top = "/content/drive/MyDrive/test_run"
s = scalesim(save_disk_space=False, verbose=True,
             config=config,
             topology=topo
             )
```

Flags:

- Verbosity
- Suppress trace file generation

Inputs

Step 3: Launch the simulation

```
run_scale()
```

```
s.run_scale(top_path=top)

====================================================
****************** SCALE SIM **********************
====================================================
Array Size:     32x32
SRAM IFMAP (kB):        64
SRAM Filter (kB):       64
SRAM OFMAP (kB):        64
Dataflow:       Weight Stationary
CSV file path:  /content/drive/MyDrive/scalesim_resources/topologies/conv_nets/alexnet_part.csv
Number of Remote Memory Banks:  1
Bandwidth:      10
Working in USE USER BANDWIDTH mode.
====================================================

Running Layer 0
100%|██████████| 112284/112284 [01:19<00:00, 1412.54it/s]
Compute cycles: 439609
Stall cycles: 327326
Overall utilization: 23.42%
Mapping efficiency: 94.53%
Average IFMAP DRAM BW: 9.997 words/cycle
Average Filter DRAM BW: 9.998 words/cycle
Average OFMAP DRAM BW: 7.907 words/cycle
Saving traces: Done!
************ SCALE SIM Run Complete ****************
```

1

Reported stats on screen

1 Total runtime and stalls

Step 3: Launch the simulation

```
run_scale()
```

```
s.run_scale(top_path=top)

====================================================
******************* SCALE SIM **********************
====================================================
Array Size:     32x32
SRAM IFMAP (kB):        64
SRAM Filter (kB):       64
SRAM OFMAP (kB):        64
Dataflow:       Weight Stationary
CSV file path:  /content/drive/MyDrive/scalesim_resources/topologies/conv_nets/alexnet_part.csv
Number of Remote Memory Banks:  1
Bandwidth:      10
Working in USE USER BANDWIDTH mode.
====================================================

Running Layer 0
100%|██████████| 112284/112284 [01:19<00:00, 1412.54it/s]
Compute cycles: 439609
Stall cycles: 327326
Overall utilization: 23.42%
Mapping efficiency: 94.53%
Average IFMAP DRAM BW: 9.997 words/cycle
Average Filter DRAM BW: 9.998 words/cycle
Average OFMAP DRAM BW: 7.907 words/cycle
Saving traces: Done!
************ SCALE SIM Run Complete ****************
```

2

Reported stats on screen

1 Total runtime and stalls

2 Array Utilization

Step 3: Launch the simulation

```
run_scale()
```

```
s.run_scale(top_path=top)

====================================================
******************* SCALE SIM *********************
====================================================
Array Size:      32x32
SRAM IFMAP (kB):        64
SRAM Filter (kB):       64
SRAM OFMAP (kB):        64
Dataflow:       Weight Stationary
CSV file path:  /content/drive/MyDrive/scalesim_resources/topologies/conv_nets/alexnet_part.csv
Number of Remote Memory Banks:  1
Bandwidth:      10
Working in USE USER BANDWIDTH mode.
====================================================

Running Layer 0
100%|██████████| 112284/112284 [01:19<00:00, 1412.54it/s]
Compute cycles: 439609
Stall cycles: 327326
Overall utilization: 23.42%
Mapping efficiency: 94.53%
Average IFMAP DRAM BW: 9.997 words/cycle
Average Filter DRAM BW: 9.998 words/cycle
Average OFMAP DRAM BW: 7.907 words/cycle
Saving traces: Done!
************ SCALE SIM Run Complete ****************
```

3

Reported stats on screen

1. Total runtime and stalls
2. Array Utilization
3. Interface bandwidths

Demo 1: Design space exploration

Quick demo of setting up SCALE-Sim for systolic design space exploration

Part a. SCALE-Sim to find best dataflow

Part b. Using SCALE-Sim to find best aspect ratio and dataflow

Inputs

Hardware configurations

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

Inputs

Hardware configurations

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

```
[general]
run_name = scale_sim_tutorial1_64x64_ws

[architecture_presets]
ArrayHeight:    64
ArrayWidth:     64
IfmapSramSzkB:    512
FilterSramSzkB:   512
OfmapSramSzkB:    512
IfmapOffset:    0
FilterOffset:   10000000
OfmapOffset:    20000000
Bandwidth : 0
Dataflow : ws
MemoryBanks:   1

[run_presets]
InterfaceBandwidth: CALC
```

We will change the dataflow when searching

Inputs

Workload

3 layer example network: 3x3 conv, 1x1 conv, FC layers

Layer name	IFMAP Height	IFMAP Width	Filter Height	Filter Width	Channels	Num Filter	Strides
Conv_3x3	56	56	3	3	64	64	1
Conv_1x1	56	56	1	1	64	256	1
FC	1	1	1	1	2048	1000	1

Demo 1: Part a

Compare the runtimes with three dataflows

Steps:

1. Install and initialize the tool
2. Run simulation
3. Analyze outputs

Step 1.

A. Install SCALE-Sim (optional)

```
!pip install scalesim==2.0.1
```

B. Import all the libraries

```
import math
import numpy as np
import matplotlib.pyplot as plt
```

```
from scalesim.scale_sim import scalesim
from scalesim.utilities.scalesim_report import ScalesimReport as reporter
```

Step 2.

Run simulations

```
[ ] def run_scale_sim():
        # 1. Read the topology file
        topofilename = tutorial_path + '/files/tutorial1_topofile.csv'

        # 2. Read the config file
        config_filename = tutorial_path + '/files/config/scale_config_64x64_os.cfg'

        # 3. Launch a SCALE-Sim run
        sim = scalesim( save_disk_space=True, verbose=True,
                        config=config_filename, topology=topofilename
            )

        sim.run_scale(top_path=tutorial_path + '/tutorial1_runs')
```

```
[ ] run_scale_sim()
```

1 Function that launches a single scale sim run

(Repeated for all the dataflows)

Step 2.

Run simulations

```
[ ] def run_scale_sim():
        # 1. Read the topology file
        topofilename = tutorial_path + '/files/tutorial1_topofile.csv'

        # 2. Read the config file
        config_filename = tutorial_path + '/files/config/scale_config_64x64_os.cfg'

        # 3. Launch a SCALE-Sim run
        sim = scalesim( save_disk_space=True, verbose=True,
                        config=config_filename, topology=topofilename
            )

        sim.run_scale(top_path=tutorial_path + '/tutorial1_runs')
```

```
[ ] run_scale_sim()
```
2

1 Function that launches a single scale sim run

(Repeated for all the dataflows)

2 Launch the run

Step 3. Compare the runtimes

```
def compare_dataflows():
    run_dir = tutorial_path + '/tutorial1_runs'
    os_run_name = 'scale_sim_tutorial1_64x64_os'
    ws_run_name = 'scale_sim_tutorial1_64x64_ws'
    is_run_name = 'scale_sim_tutorial1_64x64_is'

    os_rpt = reporter()
    ws_rpt = reporter()
    is_rpt = reporter()

    os_rpt.load_data(data_dir=run_dir, run_name=os_run_name)
    ws_rpt.load_data(data_dir=run_dir, run_name=ws_run_name)
    is_rpt.load_data(data_dir=run_dir, run_name=is_run_name)

    # Prepare arrays for plotting
    x_tick_lables = ['64x64_OS', '64x64_WS', '64x64_IS']
    y_legend = ['Layer0', 'Layer1', 'Layer2']

    os_runtimes = os_rpt.get_compute_cycles_all_layer()
    ws_runtimes = ws_rpt.get_compute_cycles_all_layer()
    is_runtimes = is_rpt.get_compute_cycles_all_layer()

    # Transpose y data
    all_y = np.asarray(os_runtimes).reshape((1,3))
    all_y = np.concatenate((all_y, np.asarray(ws_runtimes).reshape((1,3))), axis=0)
    all_y = np.concatenate((all_y, np.asarray(is_runtimes).reshape((1,3))), axis=0)

    y_series = np.transpose(all_y)

    plot_stacked_bar(x=x_tick_lables, y_series_np=y_series,
                     legends=y_legend, title='Runtime vs dataflow',
                     y_axis_label='Runtime (Cycles)')
```

1

1 Read the output of the previous runs

Step 3. Compare the runtimes

```python
def compare_dataflows():
    run_dir = tutorial_path + '/tutorial1_runs'
    os_run_name = 'scale_sim_tutorial1_64x64_os'
    ws_run_name = 'scale_sim_tutorial1_64x64_ws'
    is_run_name = 'scale_sim_tutorial1_64x64_is'

    os_rpt = reporter()
    ws_rpt = reporter()
    is_rpt = reporter()

    os_rpt.load_data(data_dir=run_dir, run_name=os_run_name)
    ws_rpt.load_data(data_dir=run_dir, run_name=ws_run_name)
    is_rpt.load_data(data_dir=run_dir, run_name=is_run_name)

    # Prepare arrays for plotting
    x_tick_lables = ['64x64_OS', '64x64_WS', '64x64_IS']
    y_legend = ['Layer0', 'Layer1', 'Layer2']

    os_runtimes = os_rpt.get_compute_cycles_all_layer()
    ws_runtimes = ws_rpt.get_compute_cycles_all_layer()
    is_runtimes = is_rpt.get_compute_cycles_all_layer()

    # Transpose y data
    all_y = np.asarray(os_runtimes).reshape((1,3))
    all_y = np.concatenate((all_y, np.asarray(ws_runtimes).reshape((1,3))), axis=0)
    all_y = np.concatenate((all_y, np.asarray(is_runtimes).reshape((1,3))), axis=0)

    y_series = np.transpose(all_y)

    plot_stacked_bar(x=x_tick_lables, y_series_np=y_series,
                     legends=y_legend, title='Runtime vs dataflow',
                     y_axis_label='Runtime (Cycles)')
```

2

1 Read the output of the previous runs

2 Get the runtimes for the three different runs

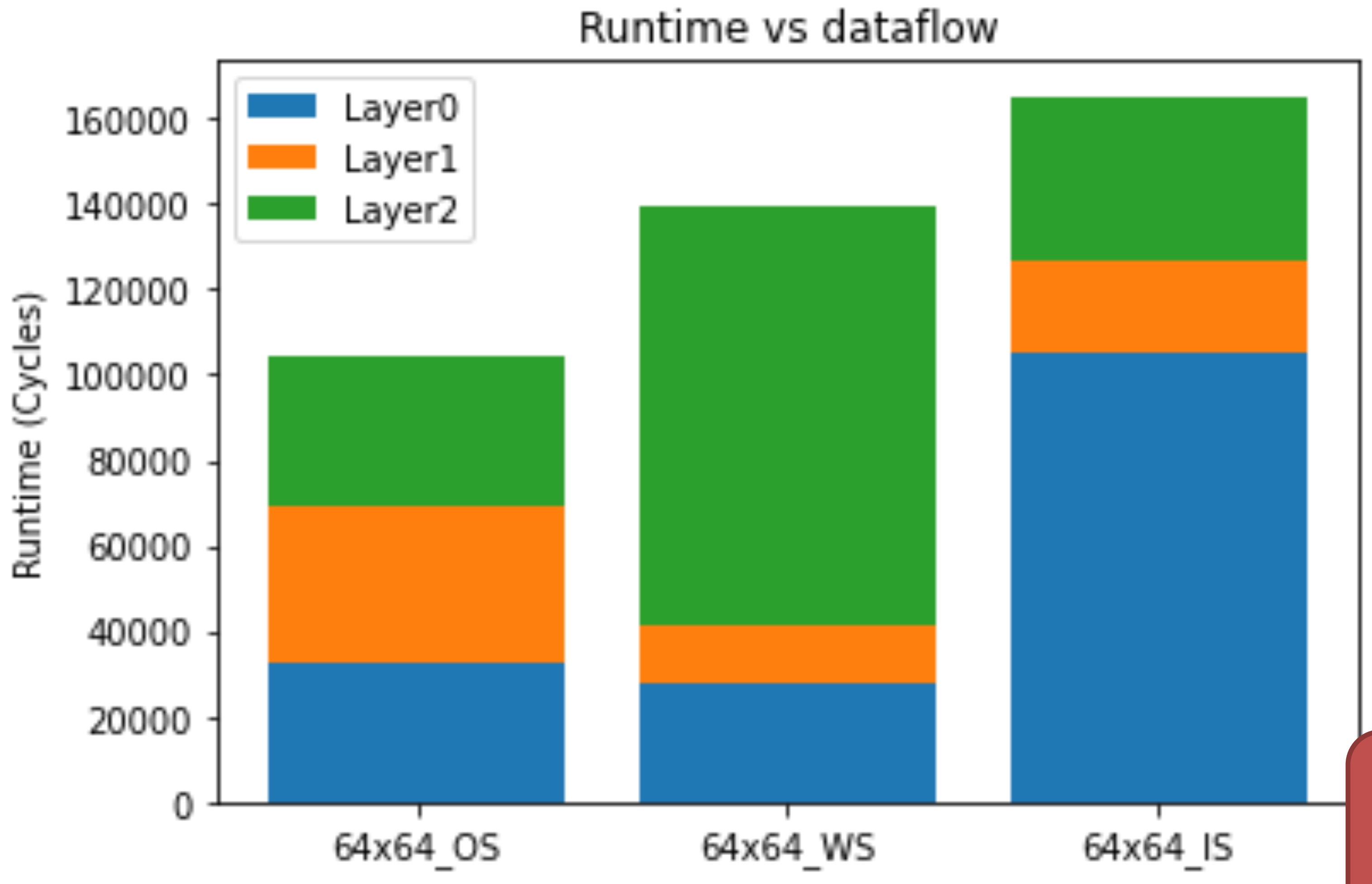
Step 3. Compare the runtimes

```python
def compare_dataflows():
    run_dir = tutorial_path + '/tutorial1_runs'
    os_run_name = 'scale_sim_tutorial1_64x64_os'
    ws_run_name = 'scale_sim_tutorial1_64x64_ws'
    is_run_name = 'scale_sim_tutorial1_64x64_is'

    os_rpt = reporter()
    ws_rpt = reporter()
    is_rpt = reporter()

    os_rpt.load_data(data_dir=run_dir, run_name=os_run_name)
    ws_rpt.load_data(data_dir=run_dir, run_name=ws_run_name)
    is_rpt.load_data(data_dir=run_dir, run_name=is_run_name)

    # Prepare arrays for plotting
    x_tick_lables = ['64x64_OS', '64x64_WS', '64x64_IS']
    y_legend = ['Layer0', 'Layer1', 'Layer2']

    os_runtimes = os_rpt.get_compute_cycles_all_layer()
    ws_runtimes = ws_rpt.get_compute_cycles_all_layer()
    is_runtimes = is_rpt.get_compute_cycles_all_layer()

    # Transpose y data
    all_y = np.asarray(os_runtimes).reshape((1,3))
    all_y = np.concatenate((all_y, np.asarray(ws_runtimes).reshape((1,3))), axis=0)
    all_y = np.concatenate((all_y, np.asarray(is_runtimes).reshape((1,3))), axis=0)

    y_series = np.transpose(all_y)

    plot_stacked_bar(x=x_tick_lables, y_series_np=y_series,
                     legends=y_legend, title='Runtime vs dataflow',
                     y_axis_label='Runtime (Cycles)')
```

3

1. Read the output of the previous runs
2. Get the runtimes for the three different runs
3. Plot the data

Analysis



Observations

1. OS stationary wins overall

2. However choice of dataflow is tightly coupled with layer dimensions

Detailed insights about computing behavior is generated with no change in code

Demo 1. Part B

Compare the effect of aspect ratio and dataflows

Additional Steps:

1. Run simulation with different configs
2. Analyze outputs

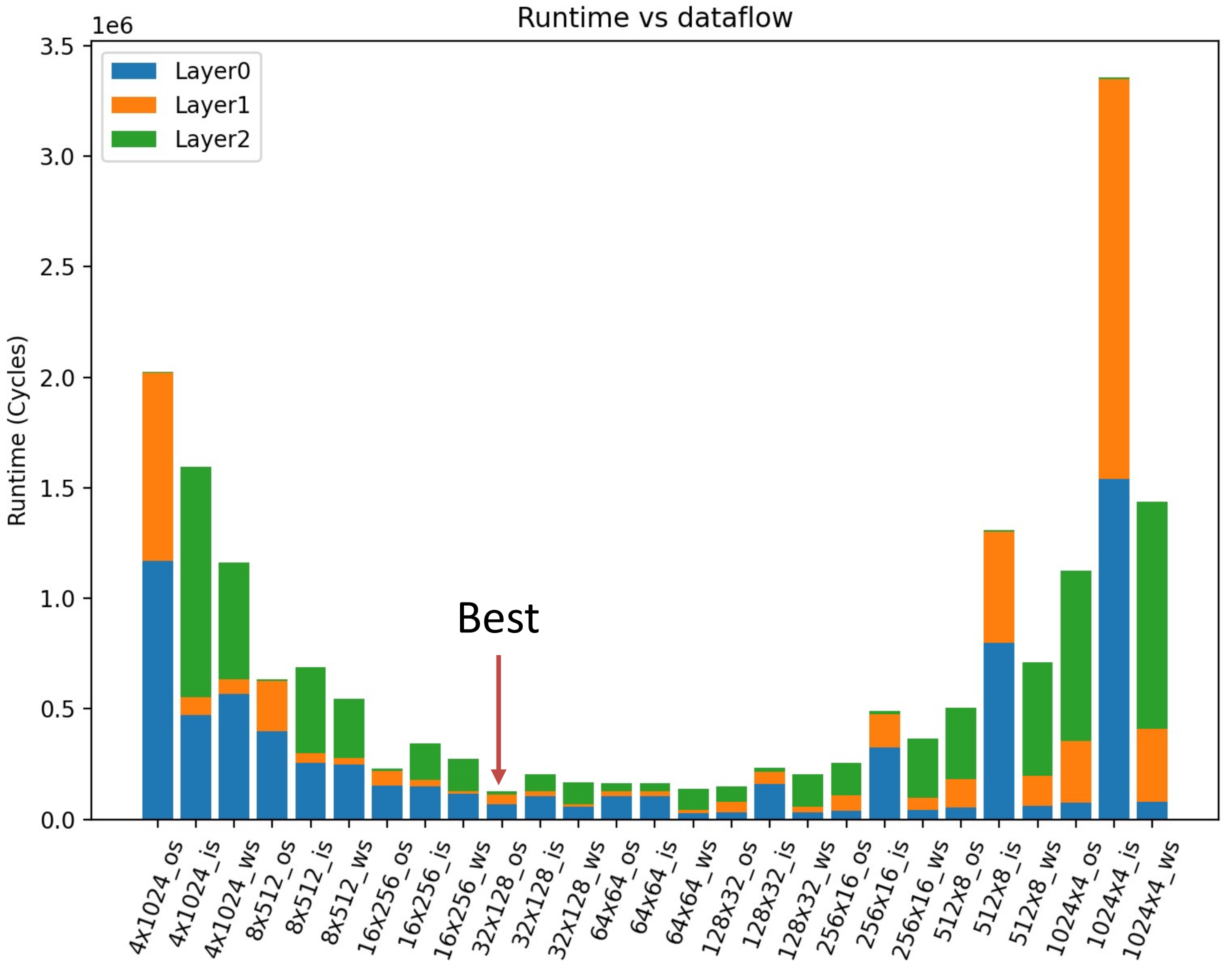
Variable aspect ratio simulations

```python
def compare_aspect_ratios():
    run_dir = tutorial_path + '/tutorial1_runs'
    x_tick_labels = []
    all_y = np.zeros((1,1))
    data_valid = False
    y_legend = ['Layer0', 'Layer1', 'Layer2']

    for rpow in range(2, 11):
        rows = int(math.pow(2, rpow))
        cols = int(round(2 ** 12 / rows))

        for df in ['os', 'is', 'ws']:
            run_name = 'scale_sim_tutorial1_' + str(rows) \
                        + 'x' + str(cols) + '_' + str(df)
            x_tick_labels += [str(rows) + 'x' + str(cols) + '_' + str(df)]
            rpt = reporter()
            rpt.load_data(data_dir=run_dir, run_name=run_name)
            runtimes = rpt.get_compute_cycles_all_layer()

            if not data_valid:
                all_y = np.asarray(runtimes).reshape((1,3))
                data_valid = True
            else:
                all_y = np.concatenate((all_y, np.asarray(runtimes).reshape((1, 3))), axis=0)

    y_series = np.transpose(all_y)

    plot_stacked_bar(x=x_tick_labels, y_series_np=y_series, legends=y_legend,
                     title='Runtime vs dataflow',
                     y_axis_label='Runtime (Cycles)')
```

Due to time constraints, we will use pre-generated data

We will use the compare_aspect_ratios() method to read and plot the runtimes

Analysis



Observations

1. 16x256 os has the best runtime, which is a skewed array

2. Like dataflow, the choice of aspect ratio is also workload dependent