ACCELERATOR BASED PROGRAMMING
UPPSALA UNIVERSITY
FALL 2023

ASSIGNMENT 3: A TENSORFLOW EXAMPLE

Due: September 28, 2023 Upload a report on Studium. Upload your code and report on Studium

1. Problem description. In this exercise, we develop a naive implementation of Conway's Game of Life using the GPU in TensorFlow. There are much more efficient algorithms that use the presence of reoccurring patterns within the Life board to speed up the processing, which are not considered here.

The rules of Game of Life are:

- Each cell has 8 neighbors, i.e., the 8 adjacent cells in each direction (including diagonals). All behavior is defined from the current state of a cell and its neighbors.
- Each cell can take two values, `1/True` or `0/False`. A live cell is a cell containing `1` , a dead cell contains `0`.
- In each iteration i of the game, all cells are updated based on the state of the cell and its neighbors in the previous iteration $i-1$. It does not matter which neighbors are turning dead/live during the current iteration i, making all operations independent.
- Any live cell with two or three neighbors survives, all other cells die (less than two neighbors: underpopulation, more than three neighbors: overpopulation).
- Any dead cell with exactly three live neighbors becomes a live cell, all other dead cells stay dead.

2. Coding environment. We will solve the problem with TensorFlow through Python. We are using a small auxiliary script `lifereader.py` (available through Studium). Make sure you have Python and TensorFlow available on your UPPMAX account as explained in exercise on Studium.

The Python code (also available as a `assignment3_template.py` on Studium) starts with importing the following modules:

```
import matplotlib.pyplot as plt
import lifereader
import numpy as np
import tensorflow as tf
import math
import timeit
```

Next, we download a zip file with lots of game of life patterns, and try at least one of them in our code. Specifically, download lifep.zip from Alan Hensel's page (also available form Studium). On an UPPMAX machine, you can download and unzip this file with the following commands from the terminal window:

```
wget http://www.ibiblio.org/lifepatterns/lifep.zip
unzip lifep.zip
```

Back in the Python code, we try loading one file with the lifereader.py script:

```
board = lifereader.readlife('files/BREEDER3.LIF', 2048)
```

The number 2048 indicates the resolution of the board in pixels in each direction. We can inspect the figure e.g. with the following two plot commands for a 200 × 200 pixel section in the center and the full panel, respectively:

```
plt.figure(figsize=(20,20))
plotstart = 924
plotend = 1124
plt.imshow(board[plotstart:plotend,plotstart:plotend])

plt.figure(figsize=(20,20))
plt.imshow(board)
```

Next, we create a tensor in TensorFlow, in this case with the `float16` type:

```
boardtf = tf.cast(board, dtype=tf.float16)
```

Next, we define the actual TensorFlow function, to be completed for Task 1 below:

```
@tf.function
def runlife(board, iters):
        # Init work

        for i in range(iters):
        # In each iteration, compute two bool tensors
        # 'survive' and 'born': TODO

        # Then, update the board by keeping these tensors
        board = tf.cast(tf.logical_or(survive, born), board.dtype)

# Finalize
return board
```

Finally, we can plot the final result and some statistics about the number of live cells:

```
tic = timeit.default_timer()
boardresult = runlife(boardtf, 1000);
toc = timeit.default_timer();
print("Compute time: " + str(toc - tic))
result = np.cast[np.int32](boardresult);
print("Cells alive at start: " + str(np.count_nonzero(board)))
print("Cells alive at end: " + str(np.count_nonzero(result)))
print(np.count_nonzero(result))
plt.figure(figsize=(20,20))
plt.imshow(result[plotstart:plotend,plotstart:plotend])
```

3. Tasks.

1. Implement the work per iteration in the part marked as TODO in the code. Use built-in TensorFlow functions as much as possible, rather than doing your own steps. The main step is to compute the number of neighbors. for which you can use the function tf.nn.conv 2d, see the documentation here:
   https://www.tensorflow.org/api_docs/python/tf/nn/conv2d
   with a 3×3 filter. Try to figure out a possible filter. You can develop both on UPPMAX or locally, as long as you have Python and TensorFlow installed. To verify the correctness of your code, the number of live cells after 1000 iterations should be 2658, compared to 603 live cells at the start.

2. Report the compute time of the GPU code on UPPMAX

3. Disable the GPU by adding the following two lines to the start of the code, before running the TensorFlow functions:
   `tf.config.set_visible_devices([], 'GPU')`
   `tf.debugging.set_log_device_placement(True)`

   Compare the run time with GPU support enabled and disabled, re- spectively. (Note that the run time on the CPU can easily reach 10 minutes or more on an UPPMAX CPU node, so plan which setting to run.)

4. Try to switch from tf.float16 to a different data type. Try which of `tf.bool, tf.uint8, tf.int32, tf.float32, tf.bfloat16` are avail able on UPPMAX and record the run time with at least three of them.