Homework 6

Assignments: All parts are to be solved individually (turned in electronically, no ZIP files or tarballs, written parts in ASCII text, NO Word, postscript, PDF etc. permitted unless explicitly stated). All filenames must be exactly what is requested, capitalization included.

Please use the ARC cluster for this assignment. All programs have to be written in JAVA/C, translated with javac/mpicc/gcc and turned in with a corresponding Makefile.

1. (50 points) Create an MPI version of the TFIDF code using the TFIDF calculations from the last 2 HWs.

   Requirements:

   - The root rank will be a "master" and distribute work to the other ranks which will be "workers".
   - Each "worker" rank will be responsible for computing the TFIDF of each word in the input files assigned to them by the "master" rank.
   - The root rank should evenly distribute responsibility for input files to each "worker" rank.
   - You must handle the case where there are many input files that are not evenly divisible by the number of "worker" ranks. There will never be less than 3 input files or 4 MPI ranks.

   The example input only has 3 files, so if we have 4 MPI ranks, there will be 1 "master" rank and 3 "worker" ranks. Each "worker" rank would only be responsible for one input file in this case.

   A serial version is provided:

   ```
   tar xvf TFIDF.tar.gz
   make
   ./TFIDF
   mv output.txt serial_output.txt
   cat serial_output.txt
   ```

   I will be grading your MPI version by comparing your output to the serial code output. I will use the given **Makefile** to compile your code. I will run the following script to grade your code.

   ```
   srun -N4 -popteron --pty /bin/bash
   make
   mpirun -np 4 ./TFIDF
   diff serial_output.txt output.txt
   ```

   In your **p1.README**:
   - Describe your implementation step-by-step. This should include descriptions of what MPI messages get sent/received by which rank, and in what order.

- Describe how you could add more parallelism to your code so that all of the processors on each MPI node are used instead of only one processor per MPI node.
- Compare your MPI implementation to the previous MapReduce and Spark implementations of TFIDF.

Turn in **TFIDF.c** and **p1.README**.

2. (Group problem, 25 points) Familiarize yourself with the Tensorflow tutorial. This tutorial implements something similar to the Lake code. You are given `lake.py` based off of this tutorial (but with a few trivial changes).

   To run this code, you need to input the grid size (`N`), the number of pebbles (`npebs`), and the number of iterations of the main loop (`num_iter`). The output will be an image named `lake_py.jpg`.

   ```
   srun -pgtx480 -n16 --pty /bin/bash
   pip2 install --user Pillow
   ./lake.py N npebs num_iter
   ```

   Note: Tensorflow can only use a GPU with capability 3.0. We do not have many of these on the ARC cluster. We will use the GTX480, which is capability 2.0. This means that Tensorflow will execute on the CPU cores of the node.

   There are 2 stencils provided in the code, a 5-point stencil and a 9-point stencil. Given the following pseudocode, you should be able to see how these stencils are constructed in python.

   - 5-point pseudocode:

     ```
     new_pixel_value = 1*(WEST + EAST + NORTH + SOUTH) +
                      -4*(old_pixel_value)
     ```

   - 9-point pseudocode:

     ```
     new_pixel_value = 1.00*(WEST + EAST + NORTH + SOUTH) +
                       0.25*(NORTHWEST + NORTHEAST + SOUTHWEST + SOUTHEAST) +
                       -5.0*(old_pixel_value)
     ```

   Your tasks:

   - Create a 13-point stencil **DIFFERENT** from HW2, and use this stencil for the rest of the assignment.
     - 13-point pseudocode:

       ```
       new_pixel_value = 1.000*(WEST + EAST + NORTH + SOUTH) +
                         0.250*(NORTHNORTH + EASTEAST + SOUTHSOUTH + WESTWEST) +
                         0.125*(WESTWESTWEST + EASTEASTEAST + NORTHNORTHNORTH + SOUTHSOUTHSOUTH) +
                         -5.50*(old_pixel_value)
       ```

The stencil is in the shape of a cross. The given python file outputs a **jpg**, but you would have to change the output to the same format as output by the previous homeworks. Your python code should output **lake_c.dat**. You need to make sure that your output file goes through the given gnuplot file to create a valid output image. Your output will look different because of the choice of stencil.

- Compare the execution time of your `lake.py` against [`lake.o`]() using the parameters `N=512`, `npebs=16`, `num_iter=400`.
  - `lake.o` is a binary of the Lake code (written in C) that uses OpenMP to run across all CPU cores, similar to Tensorflow
  - `lake.o` takes the same command line arguments as `lake.py`
  - `lake.o` will output `lake_c.dat`. View this as a PNG using [`heatmap.plot`]()

- In your **p2.README**:
  - Provide possible explanations for the difference in execution times. Test other parameter configurations to see which of the three input parameters affects execution time the most. Be sure to include the parameter values and execution time for each test case you used.

Turn in: **lake.py** and **p2.README**

3. (Group problem, 25 points)

Familiarize yourself with Horovod, which implements multi-node/multi-GPU Tensorflow via MPI. Horovod was designed for use with neural networks which don't require a send/receive abstraction, only a broadcast abstraction. We will emulate a send/receive by using broadcast with only 2 ranks. An example can be found here. Run this example using:

```
srun -pgtx480 -N2 -n32 --pty /bin/bash
pip2 install --user --no-cache-dir horovod==0.9.7
mpirun -np 2 ./example.py
```

This example creates buffers of 10 elements. Rank 0's send buffer is initialized with even numbers, and rank 1's send buffer is initialized with odd numbers. The receive buffers for both ranks are initialized to zeros. Rank 0 sends it's even buffer to rank 1, and rank 1 sends it's odd buffer to rank 0.

Your tasks:

- Create `lake-horo.py` by adding Horovod to `lake.py` in order to support multiple nodes/GPUs.
  - Your implementation will have only two ranks, a top and a bottom.
  - Each rank will have it's own randomized NxN grid of pebbles.
  - The bottom rank will communicate its top row(s) to the top rank's bottom rows.
  - The top rank will communicate its bottom row(s) to the bottom rank's top rows.
  - Each rank will output a NxN image. Stacking the images will produce the final result.

- Compare the execution time of your `lake-horo.py` against your `lake.py` using the parameters `N=512`, `npebs=40`, `num_iter=400`.

- As with the previous problem, you have to change the code to output lake_c_0.dat and lake_c_1.dat rather than lake_py_0.jpg and lake_py_1.jpg. Modify the gnuplot file to visualize your outputs.

- In your **p3.README**:
    - Provide possible explanations for the difference in execution times. Make sure to include each execution time in your comparison.

Turn in: **lake-horo.py** and **p3.README**

Peer evaluation: Each group member has to submit a peer evaluation form.

What to turn in for programming assignments:

- commented program(s) as source code, comments count 15% of the points (see class policy on guidelines on comments)
- Makefiles (if required)
- test programs as source (and input files, if required)
- README (documentation to outline solution and list commands to install/execute)
- in each file, include the following information as a comment at the top of the file, where "username" is your unity login name and the single author is the person who wrote this file:

```
Single Author info:

username FirstName MiddleInitial LastName
```

How to turn in:

Use the "Submit Homework" link on the course web page. Please upload all files individually (no zip/tar balls).

Remember: If you submit a file for the second time, it will overwrite the original file.