Homework 4

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 C, translated with mpicc/gcc and turned in with a corresponding Makefile.

1. (50 points, individual problem) Write a program that computes a TF-IDF value for each word in a corpus of documents using MapReduce.

   We need to "install" hadoop into your directory. First, append to your ~/.bashrc: `module load java`. Then, create a directory for p1. From that directory, run:

   ```
   srun -N4 -popteron --pty /bin/bash
   source hadoop-setup.sh &> setup_output.txt
   ```

   Every time you do a new `srun`, you should re-run the hadoop-setup script! Always make sure to use `source` to run it, otherwise the environment variables that the script sets will not get picked up by your current shell.

   Inspect the script's output in **setup_output.txt**. (Note: It will be extremely helpful to turn **OFF** word/line wrapping in your preferred text viewer.) It is okay to ignore any " `failed to create symbolic link: File exists`" error messages. There will be a lot of INFO and WARN statements and these are fine too. You should see a `NameNode` being started with the `-format` argument and then this `NameNode` is shut down. Then, you should see a `NameNode` being started on the ARC node you got when you ran `srun`. Then, N `DataNodes` should be started, one for each ARC node you reserved when you ran `srun`. Finally, you should see a `SecondaryNameNode` being started.

   You shoule **NOT** see any of these name/data/secondary nodes being shut down! You should **NOT** see any other errors. To check to see if everything was set up correctly, run:

   ```
   hdfs dfs -ls /user
   ```

   This should print out something like:

   ```
   Found 1 items
   drwxr-xr-x   - UNITYID supergroup          0 TIMESTAMP /user/UNITYID
   ```

   The above hadoop-setup script creates a "hadoop" directory. It then links common hadoop files to this directory and sets some environment variables. Next, it creates some personalized files in the `hadoop/etc/hadoop` directory, which contain your unity ID and the nodes you currently have reserved (this is why you have to re-run the script for every new srun). These files tell Hadoop where to store the Hadoop Distributed File System (HDFS). Temporary folders on each node are also created to hold the HDFS data. Finally, we configure a blank HDFS (starting the `NameNode` with `-format` argument), start the HDFS (starting the name/data/secondary nodes), and create a `/user/UNITY-ID` directory inside the HDFS (the one from the `hdfs dfs -ls` command).

   With hadoop successfully installed, download the TFIDF skeleton code along with sample inputs/outputs.

   ```
   tar xvf TFIDF.tar
   ```

   Fill in the **TFIDF.java** file with your own code.
   - The input to TFIDF.java will be a directory similar to the "input" directory in TFIDF.tar
   - You will accomplish the TFIDF calculation in 3 steps (jobs).
     - The first job (Word Count) will count the number of times each word appears in each document.
     - The second job (Document Size) will count the total number of words in each document. **Calculate logarithmic IDF = log(numDocs/numDocsWithWord). Why would you use log?**
     - The third job (TFIDF) will count the number of total documents, the number of documents containing each word, and finally calculate the TFIDF value for each word.
   - In between each job, a file is written containing of all the (key,value) pairs for that step.
     - Use these files to debug your code!
     - The files will be in the HDFS `/user/UNITYID/output` directory
     - Make sure to use the input and output paths that are set for you! If you don't, the grading script will not be able to find your output and you will get a zero!
   - The comments at the top of each Map and Reduce class explain the expected input and output (key,value) pairs for that function. You should follow this closely!
   - Before performing operations on (key,value) pairs, it may be helpful to first print the "key" and "value" parameters of each function to see what you are working with.
   - For `Mapper` and `Reducer`, the angle brackets are the datatypes of `< input key, input value, output key, output value>`
     - These should match the datatypes of the "key" and "value" parameters in the respective `map()` and `reduce()` functions, and also the parameters of the call to `context.write()`.
     - You should not need to change the types that are given (you can if you want).
   - You should not need to change any of the provided skeleton code (you can if you want). You should only need to add to it.
   - You should not need to add any additional files for objects, classes, etc. If your implementation requires additional files, make sure you submit them.

   Before running your TFIDF code, you must copy the input to the HDFS (substitute `UNITYID` with your unity id):

   ```
   hdfs dfs -put input /user/UNITYID/input
   ```

   Compile and run your TFIDF code:

   ```
   javac TFIDF.java
   jar cf TFIDF.jar TFIDF*.class
   hadoop jar TFIDF.jar TFIDF input &> hadoop_output.txt
   rm -rf output
   hdfs dfs -get /user/MY-UNITY-ID/output .
   ```

   The **hadoop_output.txt** file will contain the hadoop output. Inspect this output to find any runtime errors/exceptions. As you are developing your code, you can repeatedly run the above five commands to compile/run your code and get the output.

   When you are done, before releasing your reserved nodes, run the following to shut down the hdfs file system:

   ```
   hadoop/sbin/stop-dfs.sh
   ```

   Hints:

   - The output for the sample inputs should be **EXACTLY** the same as the sample output, including the directory structure and directory/file names. If you use the Path's given in the code, there should be no issues.
   - I will be grading your code using different inputs than what are provided
   - TFIDF overview
   - MapReduce Tutorial

   In your **p1.README**, explain how you implemented each of the steps of your TFIDF algorithm.

   Turn in **p1.README** and **TFIDF.java**;

2. (25 points, group problem) Inspect the LULESH 1.0 code using mpiP for the following configurations of (nodes X, OpenMP threads T, MPI tasks P, Problem Size S):

- A: (1, 8, 8, 45) -- Combination of MPI and OpenMP
- B: Same as A but without core binding
- C: (4, 0, 8, 60) -- MPI Only
- D: Same as C but without core binding

Run the following commands to compile:

```
srun -N X -n P -popteron --pty /bin/bash (Figure out X and P for A/B/C/D)
tar xvzf lulesh.tar.gz
Link LULESH with mpiP by editing the Makefile and append to LDFLAGS and LDFLAGS_OPENMP: -L/opt/ohpc/pub/libs/gnu/mvapich2/mpiP/3.4.1/lib -lmpiP -lbfd -liberty
make MPI_OPENMP (For enabling both MPI and OpenMP)
make MPI_ONLY (For enabling only MPI)
export OMP_NUM_THREADS=T (For configurations A and B)
mpirun -np P -bind-to core ./lulesh S (Here S is problem size)
```

We use

```
-bind-to core
```

option to bind threads to specific cores (to prevent migration between cores). For configurations B and D, remove this flag.

Hints:

- Do a

  ```
  make clean
  ```

  when switching between MPI_ONLY and MPI_OPENMP.
- You can ignore warnings during runtime with regards to affinity.
- You need to srun for different configurations, which will clear the environment variables. Make sure to export them again.

Evaluate the elapsed time for each of the 4 configurations, and include the timings in **p2.README**. Discuss/explain your findings in **p2.README**.

The output for mpiP will be in a file specified at the end of the run. For each configuration above, report the % of time spent in MPI, and any other intersting thing you note for any rank. Identify the most expensive MPI call and report its App% and MPI% time relative to the overall respective time. What differences are there between configurations? What are the limits to parallelization? Include this in **p2.README**.

Turn in **p2.README**.

3. (25 points, group problem) Your task is to determine the effect of frequency scaling on the LULESH code's:
   A: wall-clock time
   B: power consumption
   C: stall cycles

Before you start, here is quick lesson about Frequency Scaling and Power Monitoring on ARC (which can also be found on the ARC webpage). I will also talk quickly about Stall Cycles and how we will count them.

FREQUENCY SCALING (DVFS)
On the ARC cluster, you are able to change the clock frequency for all of the cores in a node. The `cpupower frequency-info` command will show you the "current CPU frequency", and the "available frequency steps", which is a list of frequencies at which that the cores can be set to operate. There is also a "current policy" and the "available cpufreq governors", which is a list of policies the cores can abide by.

The default policy is called "ondemand". This policy runs the cores at the lowest setting (800MHz for Opteron) and will increase the clock frequency dynamically depending on the needs of the program that is running. We must change this policy to "userspace" so that the cores will be bound to the frequency at which we tell them to operate. Once we change the policy, we can set the frequency to one of the "available frequency steps". The commands to do this are as follows:

```
sudo cpupower frequency-set -g userspace
sudo cpupower frequency-set -f 1200Mhz
```

You can check that the settings were updated by running `cpupower frequency-info` again. These settings apply to all cores within the node. It is **VERY IMPORTANT** that before releasing a node you reset the power settings back to "ondemand" by running `sudo cpupower frequency-set -g ondemand`. Otherwise, all future users of that node will be operating under an unexpected frequency!

POWER MONITORING
Sets of three compute nodes share a power meter. In each set of three, the lowest numbered node has the meter attached (either on the serial port or via USB), but the meter measures the combined power from all three nodes. Because of this, when you reserve nodes for this problem, you should reserve ALL three nodes that are a part of the set. You should run something like:

```
srun -popteron -wc[X,Y,Z] --pty /bin/bash
```

where X,Y,Z are three available (see `squeue`) nodes with a common meter attached. See this power wiring diagram to identify which nodes belong to a set. The diagram also indicates if a meter uses serial or USB for a given node.

Every 1 second, the power meter measures the combined power consumption of all three nodes. To print out these values (one per second), ensure that you are on the lowest numbered node of the set (the one with the power meter attached) and run:

```
mlogger -p 0 -o         #For serial meters
```

OR

```
wattsup ttyUSB0 watts   #For USB meters
```

You can stop the printing by doing a `Ctrl-C`. You should monitor power on one node while running the LULESH code on a different node in the set.

STALL CYCLES
Often times a core that is waiting on a memory request will issue stall cycles. A stall cycle is issued every few clock cycles until the memory request is ready. Reducing stall cycles means reducing power usage. We will use PAPI to count stall cycles that are incurred by our LULESH program. For more information about high level PAPI functions click here.

INSTRUCTIONS

- Reserve 3 nodes with a common power meter
- Remember to request 8 MPI tasks from srun (-n 8), which is required for lulesh.
- Download and unpack LULESH as before but exclude mpiP this time
- Edit the LULESH **Makefile** and append to **CXXFLAGS**: `-I${PAPI_INC}`
- Edit the LULESH **Makefile** and append to **LDFLAGS_OPENMP**: `-L${PAPI_LIB} -lpapi`
- Modify the **main()** function in **luleshMPI_OMP.cc** to measure the PAPI_RES_STL and PAPI_STL_ICY event (read about them to figure out what they mean) per MPI task and print them out in the end.
- Here is an example code to get you started with using PAPI counters.
- Open a new terminal window and ssh to one of your nodes that doesn't have the power meter attached (you will be setting the frequency and

running LULESH on this node)
- Load the PAPI module on ARC by running `module load papi`
- Run `make MPI_OPENMP` to compile the LULESH code with PAPI counters
- For each frequency in the "available frequency steps":
  - Set the node to operate on that frequency
  - Start the power monitoring in the original terminal window (it should be the node with the lowest number which has the power meter attached)
  - Run the following configuration using MPI and OpenMP (with core binding)

```
export OMP_NUM_THREADS=8
mpirun -np 8 -bind-to core ./lulesh 70
```

  - Stop the power monitoring and record the average of the power readings (watts)
  - Record the "Elapsed time" from LULESH
  - Record the 8 rank sum of both stall cycle counters from PAPI
- REMEMBER TO RESET THE NODE BACK TO THE ORIGINAL "ONDEMAND" SETTING!
- Create separate bar plots of these four metrics and combine them into one **plots.png**.
- Your **plots.png** should look something like this. (Note: These are not actual values, but the correct values are on a similar scale)

In your **p3.README** explain how scaling the frequency affects time, power, and stall counts. Be sure to discuss why these results make sense.

Turn in your modified **luleshMPI_OMP.cc**, **p3.README**, and **plots.png**

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.