Prog. HPC: Homework 4

CS 4080/7080

March 15, 2018

DUE DATES

CODE: 11:24 PM Thursday, 5 Apr 2018
REPORT: 11:24 PM Monday, 10 Apr 2018

Topic: Parallel Programming with Message Passing Interface (MPI)

The goal of this homework is to:

- develop strategies to multi-node parallelize your solution to a data processing algorithm,
- understand the use of Message Passing Interface (MPI);
- implement a map-reduce parallel algorithm concept;
- further refine your skills writing an analysis report on your implemented solution.

You wil develop a parallel approach to build and inverted file index for a folder which contains a collection of documents.

Problem: Multi-Document Index Building (Inverted Files)

See: http://en.wikipedia.org/wiki/Inverted_index

On **tc**, the following folder contains a collection of text files similar to the sample used for the MPI example.

`/data/scottgs/book`

Your code will scan the directory and process all the files. Your processing should build an inverted file that indexes all the words that occur in the files. Your code will be invoked with a directory name and the output index file name, e.g.,

```
./hw4 /data/scottgs/book myindex
```

The result of the above command would read the *book* folder, then write the output file *myindex*.

Input Format Each file in the directory is a sub-book, with one chapter:verse per line. The format follows:

```
<Title area, possibly more than one line>

<chapter>:<verse>: <String of Text>
.
.
.
<chapter>:<verse>: <String of Text>
```

Please reuse code from sample file **word_count.cpp**, in the *mpi_tc.tgz* archive, to handle individual file parsing and see examples of message passing. Note the conversion of chapter and verse into an integer for message passing. This *chapter* and *verse* will need to be converted back into the format *chapter:verse* during output (described below).

NOTE: Please monitor the course web site and Slack for code fragments. I will post directory scanning code in the near future.

Output Format The output will be formated as shown below. Remember, this output is supposed to be written to a file, not sent to standard out with the timing and logging information.

```
wordB=file1,chp:vsX; file1,chp:vsY; file2chp:vsW;...;fileN,chp:vsZ
wordC=file6,chp:vsQ; file8,chp:vsR;...;fileN,chp:vsT
.
.
.
wordZZZZ=file1,chp:vsI; file9,chp:vsJ;...;fileN,chp:vsK; fileN,chp:vsL
```

In the example above, *file* is the short-name of the file relative to the directory that was passed in on the command line, e.g., *genesis.txt*. Also, *line* should be in the *chapter:verse* format of the input, not just a numerical line number. Your code must write the inverted file to a text file in alpha-numeric order of the indexed terms.

IMPORTANT NOTE: Please adhere strictly to the specified output format, as I will be using an automated parser of the output file to verify its correctness.

Experimentation

You will run timing experiments using 1, 3, 5, 7, 9, and 11 workers. Your experiments will be to track the following times

- Total file read time, accumulated for all files in the directory
- Total time building the in-memory inverted file
- Total time writing the output file
- Total time for the entire program

NOTE: In the report section regarding this problem, provide timing trends of the various parallelization levels. You should also include some analysis of the relationship among the various timing statistics. Don't forget to use the **mpirun** command with the *-np* flag to control the parallelization within the *sbatch* scripts.

Implementation

This homework will be implemented on the *tc.rnet.missouri.edu* system. You must implement the solutions using *mpi C* or *mpi C++* or *mpi Fortran*. You may use any library functions and supporting libraries that are found on the system. Additionally, reasonable library requests will be accommodated.

Your implementation MUST USE MPI to parallel process the data. You may use either the OpenMPI or BoostMPI interface. You should use supporting sbatch scripts to run the experiment set with different levels of parallelization (see notes and example programs).

- See Canvas file: `directory_scanner.cpp` for sample code to parse a directory for regular files
- Please see the additional MPI Cluster documentation and examples for MPI on the **tc**.

Additional Implementation Constraints

- You must specify the build/runtime system in the submitted README file, along with usage instructions.
- You must implement the solutions using *C* or *C++* or *Fortran*.
- You CAN USE ANY LIBRARIES, such as boost mutexes or locks, or any other APIs not expressly forbidden.

Analysis Report

REMEMBER: The goal is to accelerate the building of the inverted file index All your timing discussions should be in the context of the time to build the index file, using parallelization level P.

Algorithm Describe your algorithm to build the inverted file index with distributed workers. Also detail your algorithms within the master and workers for delegation and recombination of work, and the file processing, respectively. You should provide a clear discussion of your overall algorithm in the context of how the parallel processes are accessing the dataset and computing partial solutions. Additionally, you must provide a clear, yet concise, discussion of when and how the results are merged from the parallel processes.

Timing Analysis I expect the analysis to include various graphical plots **and** tables of the data. For timing and analysis you will use the following P:

- P = 1, 3, 5, 7, 9, and 11 workers;

Offer some interesting insights and explain which portions of your code/algorithms results in the various trend characteristics. How does Ahmdahl's Law or Gustafson's Law apply to your algorithm or code performance? Is there an optimal performance point?

Lessons Learned Conclude with a discussion of the key things you have learned during this exercise in regards to multi-node programming, parallel algorithm development, conducting analysis of code, etc.

Deliverables

You will submit two files.

1. A tar-ball of a directory containing all the source code and appropriate Makefiles or CMAKE or Autotools. This should include a top level *README* file that details the simple procedure to build and run the program(s), as well as the necessary analysis. You should place all your submission material into a folder named as follows: *pawprint_hw4*. Where the *pawprint* is your actual pawprint id. To prepare your submission, you must remove any data files and object files and SLURM logs and executables from the folder. The naming convention should be: **pawprint_hw4.tgz**. Where the *pawprint* is your actual pawprint id.
   Submitted into the appropriate link in Canvas.

2. A *PDF* report providing:

   - a moderately detailed summary of your solution to implementing parallel inverted file construction (1 – 2 pages);
   - some type of visual depiction of the timing trend(s) of the parallelization as the worker count increases;
   - analysis (1-2 pages) of the timing trends;
   - and any other relevant insights you gained through the exercise.

   The report should not exceed 10 pages and should identify your *Name* and *Pawprint* at the top of the front page. The naming convention should be: **pawprint_hw4.pdf** Again, the *pawprint* is your actual pawprint id.
   Submit the report into Canvas.