CMDA 3634 Fall 2017 Homework 03

T. Warburton

October 12, 2017

You must complete the following task by 11:59pm on Friday 10/20/17.

Your write up for this homework should be presented in a LaTeX formatted PDF document. You may copy the LaTeX used to prepare this report as follows

1. Click on this link
2. Click on Menu/Copy Project.
3. Modify the HW03.tex document to respond to the following questions.
4. Remember: click the Recompile button to rebuild the document when you have made edits.
5. Remember: Change the author

Each student must individually upload the following files to the CMDA 3634 Canvas page at https://canvas.vt.edu

1. `firstnameLastnameHW03.tex` LaTeX file.
2. Any figure files to be included by `firstnameLastnameHW03.tex` file.
3. `firstnameLastnameHW03.pdf` PDF file.
4. `network.c` text file with student code.

Additionally, source code is to be pushed to a GitHub repository. See question **Q1** for instructions on submitting your code via GitHub. (This does not replace the submission of your `network.c` file to Canvas. Having code on GitHub is an additional requirement.)

Other notes:

1. A `network.c` starter code is available on the course GitHub repository. Note this program only takes one input, the data file, and prints the PageRank of all nodes. The starter code is complete with an `mpi_updatePageRank` method. If you use the starter code, you will have to conform your `mpi_networkReader` and `mpi_computeDiff` methods to cooperate with the update method.
2. Use only the data fields that are in the network struct in the starter code. Do not add additional data fields.
3. If you choose, you may refrain from using the starter code. In this case, you will also be writing your own update method. Use whatever strategy for dividing the problem and updating the PageRank you would like (though it must include MPI send and recv calls). If you choose not to use the starter code include a note in your report affirming that you have done so and include a pledge that all code is your original work. Extra Credit will be awarded as appropriate.

You must complete this assignment on your own.

170 points will be awarded for a successful completion.
Extra credit will be awarded as appropriate.

Q1 *(20 points) Setting up a GitHub repository.*

Create a GitHub repository for CMDA3634. Give Dr. Warburton (GitHub username: tcew) and William Winter (GitHub username: wmwinter) access. Create a "HW03" sub-directory and push your `network.c` source code to that sub-directory. Starting with this assignment, pushing homework source code to GitHub is a requirement.

Q2 *(30 points) Lecture 11 in class assignment.*

30 points will be awarded for attending class on October 4th and submitting "computing pi in parallel" code to GitHub. Make sure Dr. Warburton and William Winter have access to the repository the source code is in. See **Q1** for more details on GitHub requirements.

Q3 *(50 points) Distributing data to multiple processors.*

Read carefully before beginning: First we must decide how to split our problem among multiple processors. The most basic strategy is to "assign" each processor a set of nodes to maintain. To do this, we will use the modulo operator. In C, the modulo operator is %. The operations preforms division, but returns the *remainder* instead of the quotient. So $a\%b$ is "the remainder when a is divided by b".

Now we use the modulo operator to assign nodes to processors. Let i be some node, let *size* be the number of processors, and let r be some processor's rank. For this assignment, we split the nodes among the processors in the following way: If i%size == r, then node i belongs to the processor with rank r. Meanwhile, the index of that node *within* the processor it belongs to will be i/size (cast as an integer/rounded down). This effectively restarts indexing inside of each processor.

Example: To illustrate this strategy, suppose we have 5 nodes to distribute and 2 processors. Processor 0 will be assigned nodes $0, 2$, and 4 while processor 1 will get nodes 1 and 3. But inside each processor, we restart indexing. So locally on processor 0, node 0 is 0, node 2 is 1, and node 4 is 2. While in processor 1, node 1 is 0, and node 3 is 1. So the PageRank arrays in processor 0 would only have 3 entries, but the PageRank arrays in processor 1 would only have 2 entries in this scenario. This indexing structure should be followed exactly. Otherwise the supplied update method will not operate properly.

Edit the supplied `network.c` file to add the following functionality. Create a `mpi_networkReader` method. The method should involve the root processor reading the input data file and distributing pertinent data to all processors. NOTE: We are not sending all information to every node! Each processor should only have information that pertains to its own nodes. So, each processor should have source and destination information for every connection where at least one of the nodes involved belongs to that processor. Note that to save on data transfers, the code assumes connections appear in the same order on all nodes. The code uses the order the connections appear in the data file (the `.csv` file) as a master ordering. So while each processor may not have all the connections in the data file, the ones it does have appear in the same order as in the data file.

Each processor should also have information on how many outbound connections of the nodes that belong to it.
Here is an outline of what the method should look like:

```
Network mpi_networkReader(char* filename){

  int rank, size;
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
```

```
  if(rank==0){

  }
  else{

  }

}
```

Note: Your code will have to use the variables in the network struct already provided. You should not add additional variables to the network struct.

Q4 *(50 points) Computing diff by reduction.*

Computing diff requires a reduction. Complete the `mpi_computeDiff` method. Use the prototype:

```
double mpi_computeDiff(Network net){

}
```

Outline/Hints: Each process should compute the sum of square differences for its own nodes. Then, an mpi based reduction should be preformed so that each node has the sum over all nodes. Have each processor take the square root of the summed diff, and each node has an accurate normed difference between the new and old PageRank arrays. Note that it may be easier to keep organized if a separate `mpi_sumDiff` method is defined. This is not required, but is recommended. You may use class reduce/barrier code as a starting point. Code should work correctly for any number of processors (not necessarily powers of 2).

Q5 *(20 points) Visualizing message passing with Jumpshot.*

Use Jumpshot to visualize the MPI program running. Run the program with 4 processors. Take a few screen shots of the plot generated by Jumpshot and include them in the report. The images should be in the report pdf (see earlier homeworks use of `\includegraphics` to figure out how to do this). The image files should also be uploaded to canvas along with the report pdf, tex, and `network.c` file.

Extra Credit: *(Up to 20 points)*
Attend an Advanced Research Computing (ARC) tutorial. Submit a selfie of you at the session. Make it clear from the selfie that you were actually in attendance.
