ECE 420 Parallel and Distributed Programming
Lab 4: Implementing PageRank with MPI

Winter 2017

1 Introduction

PageRank is the first algorithm used by Google search engine to evaluate the popularity of webpages and rank the results from the search query. The intuition of this algorithm comes from the fact that more important webpages will have more links from other webpages. In the meantime, since more people visit these popular webpages and may click the links in these webpages, these webpages will also contribute to the popularity of other webpages they link to. In the probabilistic view, the PageRank algorithm outputs a probability distribution representing the likelihood that a person randomly clicking on links in the Internet will arrive at any particular webpage.

Suppose we have a group of webpages, represented by nodes in a graph. There will be a directed link from node A to node B if the webpage represented by node A contains a link to the webpage represented by node B. In PageRank, we can also suppose that initially, all the nodes have an *equal probability* of getting visited. Furthermore, we assume an Internet surfer currently visiting node A will randomly transfer from node A to any of the nodes that node A links to *with equal probability*.

Suppose that there are N nodes. Let vector $\vec{r}(t) = (r_1(t), \ldots, r_i(t), \ldots, r_N(t))$ denote the PageRank values of all the nodes in the t^{th} iteration, where $r_i(t)$ represents the probability that the surfer is on node i in the t^{th} iteration. Initially, we have

$$r_i(0) = \frac{1}{N} \quad i = 1, \ldots, N. \tag{1}$$

Then, according to the random surfing model described above, there is a recursive relationship between $\vec{r}(t)$ and $\vec{r}(t+1)$, i.e.,

$$r_i(t+1) = \sum_{j \in D_i} \frac{r_j(t)}{l_j}, \quad i = 1, \ldots N, \tag{2}$$

where D_i denotes the set of nodes that have a outgoing link to node i, and l_j denotes the total number of outgoing links from node j to other nodes.

In the PageRank algorithm, we make the additional assumption that an imaginary surfer who is randomly clicking on links will eventually stop clicking at some point. Specifically, there is a damping factor $d < 1$ to represent the probability that the person will continue clicking on the links in each step (in each iteration). Therefore, instead of using the relationship in (2), we usually add a damping factor to obtain the following iterative updates:

$$r_i(t+1) = (1-d) \cdot \frac{1}{N} + d \cdot \sum_{j \in D_i} \frac{r_j(t)}{l_j}, \quad i = 1, \ldots, N, \tag{3}$$

where d is empirically set to around 0.85 in most applications.

In real situation, there exists nodes that have no out going links. Those nodes will cause some problems in (3) since it brings about a zero denominator in $\frac{r_j(t)}{l_j}$. To deal with the node with no out going links, we can manually link it to every nodes (including itself) and update the l_j and D_i accordingly before the calculation.

A natural way to compute $\vec{r}(t)$ is to carry out the iterative updates in (3) for a number of iterations. This leads to the idea of the **Iterative Approach** to calculate PageRank values. We can start with the initial assignment from (1) and iteratively update $\vec{r}(t)$ according to (3). This procedure may be terminated if the difference between $\vec{r}(t+1)$ and $\vec{r}(t)$ is sufficiently small. A good measure of the difference is the relative change of $\vec{r}(t)$ in each iteration under a certain norm. Therefore, we can terminate the algorithm once the following condition holds:

$$\frac{\|\vec{r}(t+1) - \vec{r}(t)\|}{\|\vec{r}(t)\|} < \epsilon, \tag{4}$$

where ϵ is some predefined positive constant. In this lab, let us set $\epsilon = 1 \times 10^{-5}$.

2 Tasks and Requirements

Task: Implement a parallel/distributed version of the PageRank algorithm described in the previous section with MPI based on the iterative updates described above.

Implementation Requirements and Remarks:

1 Use the scripts in "Development Kit Lab 4" to generate input data (which is a graph with directed links) and save results. The results should be the PageRank values of node 0 to node $N-1$, where the indices of nodes in your output should match those provided in the input data. For data loading, since there are different ways to represent the input data (the graph), you can design and use any of your own functions to load data. Refer to the *ReadMe* file for more details.

2 Use the "double" data type to store your results (the PageRank values) in order to achieve a high accuracy and be compatible to the provided testing script.

3 Time measurements should be implemented in a proper way.

4 DO NOT pass any command line arguments to your program.

5 Your program must be able to run under different problem sizes and under different numbers of processes. *But you can always assume that the problem size is divisible by the number of processes.*

6 Test your program BOTH on a single computer AND on a cluster of computers.

7 *Optimize* the performance of your implementation as much as possible using the techniques learned in class.

8 **Important Note: We will only test the correctness and performance of your code on a single machine.** For experiments on multiple machines, you only need to write down your observations and discussions in the lab report.

Lab Report Requirements:

1. Describe your implementation clearly.
2. Compare and discuss the performance (speedup, or efficiency, or cost) of your implementation under different parameter settings. Compare the performance from the single-machine experiments with that from multiple-machine experiments. Explain your observations.
3. Answer the following *mandatory* questions at the end of your report:
   (1) Whether the performance of your program is better on a single machine or on multiple machines? What's the reason? What kind of problems will benefit from a distributed program running on multiple machines?
   (2) How did you partition your data? One node per process or multiple nodes per process? What's the best number of processes in your program? How the granularity will affect the running time of your program and why?
   (3) What communication patterns are mainly used in your program? What's the advantage of your specific choices?
4. **(Optional)** Please also think about the following *optional* questions.
   (1) Did you use non-blocking communications to overlap communication and computation? If yes, was there a benefit by using non-blocking communications?
5. Please also refer to the "Lab Report Guide" for other requirements.

Submission Instructions:

Each team is required to submit BOTH a hard copy of printed lab report to the assignment box AND the source code on eClass. The report should be submitted to the assignment box on the 2nd floor of ECERF. The code should be submitted on eClass. Please check the due date of code and report submission on eClass.

For code submission, each team is required to submit a **zip** file to eClass. The zip file should be named "StudentID-Hx.zip", where "StudentID" is the Student ID of **one** of your group members (doesn't matter which member) and "Hx" is the section (H1 or H2).

The zip file should contain the following files:

1. “readme”: a text file containing instructions on how to compile your source files;
2. “members”: a text file listing the student IDs of ALL group members, with each student ID occupying one line (put only *one* “line break” at the end of each line);
3. “Makefile”: the makefile to generate the executable file. Please ensure that your Makefile is located in the root folder of that zip file and the default “$make” command will generate the final executable program named “main”;
4. All the necessary source files to build the executable “main”;

DO NOT include the compiled data generation file, “serialtester” file, or the input/output data file.

Note: you MUST use the file names suggested above. File names are case-sensitive. You MUST generate the required zip file by directly compressing all the above files, rather than compressing a folder containing those files.

Appendix

A Marking Guideline

Code:	
Correct results under different settings:	2
Time Measurement:	1
Report:	
Clear description of implementation:	1
Testing and verification:	1
Performance discussion:	3
Questions:	2
Total:	**10**