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

Winter 2022

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 is that if many webpages have access links referring to a common target webpage, then the target webpage is popular. Since a popular webpage is likely to draw more visitors, its access links are also more likely to be clicked on, thereby further contributing to the popularity of other webpages referred through these access links. In a probabilistic view, the PageRank algorithm outputs a probability distribution representing the likelihood that a person randomly clicking through links on the Internet will arrive at a 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 an access 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 an outgoing link to node i, and l_j denotes the total number of outgoing links originating from node j.

In the PageRank algorithm, we make the additional assumption that the Internet surfer, who is randomly clicking on the 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 to click on the links at 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 set to around 0.85 in most applications.

We also note that there may exist nodes that have no outgoing links. Those nodes will cause some problems in (3) since it introduces a zero denominator in $\frac{r_j(t)}{l_j}$. To deal with nodes that have no outgoing links, we introduce a self-referencing link for each node and update the l_j and D_i accordingly before performing computation (3) (the provided data generator in the development kit has already done this for you).

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 is reached:

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

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 distributed version of the PageRank algorithm described in the previous section with MPI.

Implementation Requirements and Remarks:

1 Use the scripts in "Development Kit Lab 4" to generate input data (which is a graph with directed links), load data, 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. 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 high accuracy and ensure compatibility with the provided test script "serialtester.c".

3 Time measurements should be implemented in a proper way. Specifically, it should only measure the time required to run the PageRank algorithm. Graph loading and results saving time should not be included.

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

5 Your program must be able to be run under different problem sizes, under different numbers of processes and on different number of machines[1].

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

7 Test the performance of your program both on a single machine and on a cluster of machines. Discuss your findings under the following problem sizes: 1112 nodes (./datatrim -b 5300), 5424 nodes (-b 13000), 10000 nodes (-b 18789). For each given input size, repeat your experiment 10 times and measure the average performance.

Hint: You do need to handle the case where the number of nodes in the graph is not perfectly divisible by the number of processes. One possible design is to append some padding 0s if necessary before your core computation and remove

[1] In this lab, "machines" refers to the nodes in the VM cluster.

those padding parts when saving the results. This way, your core computation segment could be simply reused with minimal changes.

Lab Report Requirements:

1. Describe your implementation clearly.
2. Record your experimental observations using tables, charts, or graphs. For each problem size, the experiments should account for:
   (1) Running on a single machine or multiple machines (on the VM cluster),
   (2) Running with different number of processes.
3. Compare and discuss the performance (time, speedup, efficiency, or cost) of your implementations under different parameter settings. Answer the following questions in your report:
   (1) Is the performance of your program better on a single machine setup or on a multiple machines setup? What is the reason?
   (2) What is the best number of processes that should be used in your program, respective to the different problem sizes? How does the granularity affect the running time of your program and why?
   (3) How did you partition your data? How did you partition the graph among the processes?
   (4) What communication mechanisms are used in your program? What is the advantage of your specific choices in terms of communication overhead and running time?
4. Please also refer to the "Lab Report Guide" for other requirements.

Submission Instructions:

Each team is required to submit a zip file to eClass by the submission deadline. The zip file should be named "StudentID.zip", where "StudentID" is the Student ID of **one** of your group members (it does not matter which member, though this should be consistent for all submissions throughout the course).

The zip file should contain the following three folders:

1. "Code": this folder should contain all the codes necessary to compile the "main" executable, including:
   (a) "Makefile": the makefile should generate the executable "main" upon issuing the "make" command.
   (b) All the necessary source files to build the solution executable.
2. "Members": this folder should contain a single text file *named* "members.txt", listing the student IDs of ALL group members, with each student ID occupying one line.
3. "Report": this folder should contain the lab report, saved as a single PDF file named "lab_report.pdf".

DO NOT include any executables, or the input/output data file. Also, please do not include the "web-Stanford.txt" raw data file.

Note: you must use the file names suggested above. File names are case-sensitive. Please generate the required zip file by directly compressing all of the aforementioned folders, rather than compressing a parent folder containing these folders as subfolders.

A Appendix: Marking Guideline

Code (4 Marks):

- Correct results on different input sizes on a single machine
- Correct results on different input sizes on a cluster (multiple machines)
- Speedup of your best program on the VM cluster

Report (6 Marks):

- Clear description of your implementation
- Discussion of scalability issues (running on single/multiple machines)
- Discussion of granularity issues (the best number of processes)
- Partitioning, communication and other issues
- Presentation of experiment results

B Appendix: Testing on A Computer Cluster

To test your MPI program on the VM clusters, you need to follow the procedures below:

1. Log into your VM cluster. In the home directory, there is a file named *hosts* that contains IP addresses of the destination machines, on which you will test your program.
2. DELIVER your executable file and any supporting files to all the destination machines (recorded in the *hosts* file) on the SAME path.
3. Launch your program,
   $ mpirun -np 4 -f hosts ./main
   This will start 4 processes on the machines in *hosts* to run the "main" program.