ECE 420 Parallel and Distributed Programming Project: PageRank Implementation via MPI

Winter 2015

1 Introduction

PageRank is the first algorithm used by the Google search engine to evaluate the popularity of websites and rank the results from the search engine. The intuition of this algorithm comes from that the more important websites will get more links and they will also add more importance to the websites they link to. In the probability view, the PageRank algorithm outputs a probability distribution used to represent the likelihood that a person randomly clicking on links will arrive at any particular page.

Suppose we have a group of webpages, which denote as the nodes in a graph. There will be a directed link from one node to another if the corresponding webpage contains a link to the other. In PageRank, we can also suppose that initially, all the nodes have an equal probability to get visited. Furthermore, we assume that the chance that a random surfer will transfer from one node to any of the nodes it links to is equal as well.

Suppose there are N nodes. At t^{th} iteration, the probability that the surfer will be on node i is $r_i(t)$. The corresponding vector containing all the PageRank values is $\vec{r}(t) = [r_i(t)]$. Initially, we have

$$r_i(0) = \frac{1}{N} \text{ for any } i. \tag{1}$$

There is also a recursive relationship between $\vec{r}(t)$ and $\vec{r}(t+1)$:

$$r_i(t+1) = \sum_{j \in D_i} \frac{r_j(t)}{l_j}, \text{ for any } i, \tag{2}$$

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

The PageRank theory holds that an imaginary surfer who is randomly clicking on links will eventually stop clicking. The probability, at any step, that the person will continue is a damping factor d. Therefore, instead of using the relationship in (2), we usually add a damping factor to the recursive relation to obtain

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

where d is empirically to set to around 0.85 in general. What's more, we set $l_i = N - 1$ for those nodes that have no outgoing links. In this case, the surfer may choose to transfer to a random other webpage if the current webpage contains no links.

Now we present two approaches to calculate the PageRank value. A natural guess is that $\vec{r}(t)$ will become stable after a number of iterative updates. This leads to the idea of the **Iterative Approach** to calculate the PageRank value. We can start with the initial assignment from (1) and recursively update $\vec{r}(t)$ by (3). We can terminate this procedure if the change between $\vec{r}(t+1)$ and $\vec{r}(t)$ is sufficiently small, say smaller than a predefined threshold $\epsilon > 0$, i.e., we terminate the calculation, once the following holds:

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

The other approach is the **Algebraic Approach**. By letting t to ∞ in (3), we have

$$\vec{r} = \frac{1-d}{N} \cdot \vec{1} + \mathbf{B} \cdot \vec{r}, \tag{5}$$

where $\vec{1}$ is the column vector containing N ones and $\mathbf{B}$ is a matrix characterizing the transfer probability defined as

$$\mathbf{B} := (\mathbf{L}^{-1} \cdot \mathbf{A})^T, \tag{6}$$

where $\mathbf{L}$ is a diagonal matrix with l_i (the number of outgoing links for node i) on the main diagonal (zero elements elsewhere), and $\mathbf{A}$ is the adjacent matrix in which each element A_{ij} is 1 if the link from i to j exists and 0 otherwise. From (5), we get

$$(\mathbf{I} - (\mathbf{L}^{-1} \cdot \mathbf{A})^T) \cdot \vec{r} = \frac{1-d}{N} \cdot \vec{1}, \tag{7}$$

which indicates that we can get the PageRank values by solving a linear equation of either (7) or (5).

2 Task and Requirements

Task: Implement a parallel version of the PageRank Algorithm via MPI using *either* the Iterative Approach *or* the Algebraic Approach. Choose one approach and optimize the performance of your implementation.

Implementation Requirements:

- An input dataset is provided for you to calculate its PageRank. Your program should save the output in a file with the name "data_output" in which every row contains two elements: the first is the node ID and the second is its PageRank value. Please use "double" type data to store your calculated PageRank value.
- You need to optimize the performance your program as much as you can. You should demonstrate that you have considered multiple performance metrics, e.g., runtime, speedup, storage overhead (memory), communication overhead, computation overhead, just to name a few.
- For time measurement, please put the start point right before your data loading section and put the end point right before your data saving section (i.e., the data loading time is counted while the data saving time is excluded). Please test your program on the machines in the lab. In your report, please indicate the machine(s) you use.

Report Requirements:

The general guideline for the final project report is similar to that for the previous lab reports. The **differences** are

- Now your page limit is 6 instead of 3 (excluding all figures, tables, code, etc.).
- Please add a brief introduction section to give an overview of what you will present. This is to make your report more readable.

In the Description of Implementation section, besides clearly describing your implementation, please also show your thoughts about the choices to optimize your design. In the Performance Discussion section, besides the runtime analysis of your final design, please try to justify the optimality of your implementation by comparing its performance against some baseline design(s) or some of your early stage design(s).

Deliverables and Submission:

- The report: please hand it in the assignment box by 4:00 p.m. on April 10. Append experimental results and the printed source code to the hard copy you hand in.
- Source code: in addition to the hardcopy submitted, please upload a zipped file containing all source code through eClass in which a readme file should be included to describe how to compile and run your code. Do not include the input data file. The code should be uploaded by 4:00 p.m. on April 10 too.