New York University Tandon School of Engineering

Electrical and Computer Engineering

Project Description

(Assigned Monday 11/23, Due Friday 12/18)

Note About Project Submission*: the assignment is meant to be done in* ***teams of 2****. You are allowed to choose your own project partner. A team will be given a single grade on the project, which will be assigned to both partners.*

The project submission involves submitting your implementation and a project report by the due date. The submissions will be made on Newclasses.

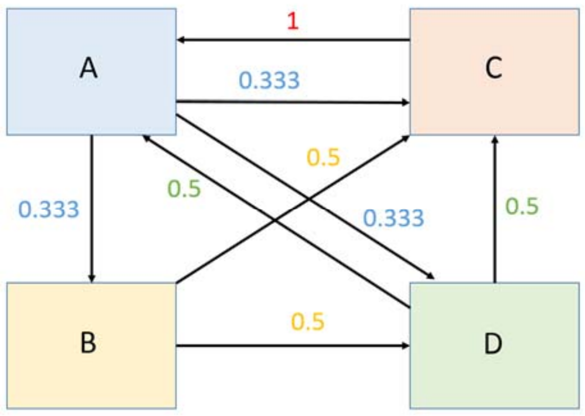
Pagerank Algorithm

The Pagerank algorithm is the cornerstone of web search. Broadly, given a graph where each node is a website and directed edges between nodes represent links from one website to another, Pagerank allocates a *score* to each website.

The score reflects a websites "importance." When you search for a keyword, the search results are returned in decreasing order of this score, that is, the higher the score, the higher its rank.

How are these scores assigned? The basic idea is as follows: a site's score depends on how many other sites link to it. For instance, many websites link to new stories on the New York Times' webpage, so the New York Times webpage (likely) has a high score.

But of course, just having a lot of websites linking to your webpage is not enough. What matters (in addition) is how important are the websites linking to your webpage? That is, if the New York Times provides a link to my webpage, it matters more than, say, the Des Moines Register (a less widely read newspaper) linking to my webpage.



$$Adj = \begin{array}{c|cccc|c} & A & B & C & D & \\ \hline & 0 & 0 & 1 & 1 & A \\ & 1 & 0 & 0 & 0 & B \\ & 1 & 1 & 0 & 1 & C \\ & 1 & 1 & 0 & 0 & D \end{array}$$

$$W = \begin{array}{c|c} 0.333 & A \\ 0.5 & B \\ 1 & C \\ 0.5 & D \end{array}$$

All of the intuition above is captured by the Pagerank algorithm in assigning scores. Consider the graph with N = 4 webpages shown above[1]. It represents 4 websites (A, B, C and D). Arrows represents links. For instance, observe that website A *links to* website B.

The graph structure is described using an adjacency matrix, *Adj*. Each row in the matrix corresponds to a website. Similarly, each column corresponds to a website. *Column i in Row j is 1 if website i links to website j*. For example, the row corresponding to website A indicates that websites C and D link to it.

Each website also has a weight stored in vector W. The weight of a website corresponds to the number of *outgoing edges* from that website. For instance, website A has three outgoing edges (links to websites B,C, and D) and therefore has a weight of 1/3.

The idea behind the weight is that a website splits its importance score equally amongst its outgoing edges.

Now, Pagerank works as follows. Let S = {s(A), s(B), s(C), s(D)} be the score of every webpage. Pagerank will start by setting the scores to be equal. That is:

$$s(A) = s(B) = s(C) = s(D) = 1/N = 0.25.$$

Starting with these scores, Pagerank iteratively updates the scores as demonstrated with the example below:

$$s(A) = (d/N) + (1-d) * (s(C)/W(C) + s(D)/W(D)).$$

Let's try and understand the equation above. First, note that d is a "magic parameter" that is an input to this algorithm. We will assume d = 0.15.

Now, note that the new score for website A depends on the sum of the (old) scores for the websites that link to it divided by their respective weights. This sum is multiplied by 1-d = 0.85.

The scores for all other websites are updated accordingly. This completes *one iteration* of Pagerank.

Pagerank keeps iterating, as shown above, until the scores converge; that is, the scores either stop changing altogether, or the change is scores is lower than some threshold, say 0.01%.

[1] This example is courtesy of http://www.math.cornell.edu/~mec/Winter2009/RalucaRemus/Lecture3/lecture3.html

Sample Implementation

To get you started, we have provided a sample implementation of the Pagerank algorithm, and a testbench that exercises it for the example shown above.

However, the sample implementation has several shortcomings:

1. It uses point-to-point communication. In the code, every website can read the score of every other website to update its own score. We have discussed in class that this type of approach does not scale well as N increases.

2. It is fully parallel. In other words, updates for every website happen in the same clock cycle. As we have discussed in class, this has a large area cost. A better option might be implement some updates in sequence, re-using the same hardware.

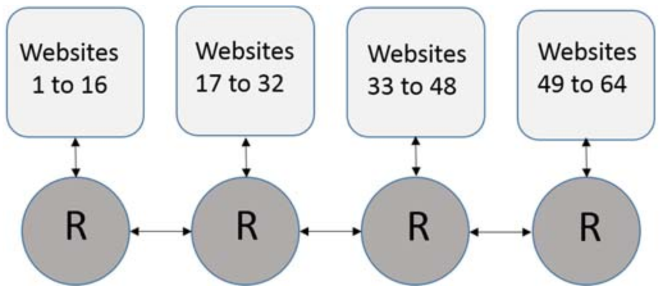
3. The implementation does not output a *ranking*. In other words, once the algorithm has converged, the final step is to sort the scores and output the "top-10" website IDs.

4. It uses a 16-bit fixed point representation for the weights and scores. Pagerank can sometimes require high accuracy for convergence; potentially require a more precise fixed point representation (more bits!) or perhaps even floating point.

5. Finally, it does not have any convergence rule --- when is the Pagerank algorithm done? A simple rule would be to run the algorithm for a fixed number of iterations or till the scores stop changing. More realistically, one might want to detect the point beyond which the change in scores is below a threshold.

What You Must Do

In order to address the shortcomings above, we want you to make the following changes.



1. NOC based implementation: implement an NOC based communication scheme instead of the point-to-point approach in the sample code. An example is shown above for N = 64.

   Here, we have split up the design into 4 processing modules, each of which deals with 16 websites. Communication between processing modules occurs over a linear NOC.

   You have several degrees of freedom here: what is the NOC topology, how many routers, what is the buffer depth of each NOC router, what is the routing and arbitration scheme etc.

   Also, you can decide the number of websites per processing modules, whether the updates happen in parallel or sequence (or a mix) in every processing module and so on.

2. The final output from your implementation must be the "ID" of the top-10 websites, and their Pagerank scores. To do this, you might have to add another "sorting" module (or modules) to the design.

3. Add a convergence rule (and make sure your design converges).
   If you are observing convergence issues, you might want to increase the precision of the fixed point representation. Also, the simplest approach would be to finish after a certain number of iterations; but is that the best?

   Along with the sample Verilog code, we have also provided Matlab code that allows you to input an adjacency matrix and determine an appropriate convergence rule.

4. Metric: your final design will be judged based on its area (A), and computation time (T) (i.e., the time taken to output the sorted "top-10" websites and their convergence scores). The computation time will be computed as the product of the clock period at which your design synthesizes, and the number of clock cycles it takes to produce an output.

   $$\text{Metric} = A^{1.5} * T$$

   Area is measured in um^2 (as reported by Cadence) and time in seconds.

5. We encourage you to be creative. You can add extra features to your design over and above the to-dos listed above. For instance; consider clock gating processing modules that have already converged.