# Project #2-A: MPJ PageRank (Due Oct. 9th, 11:59 pm)

1. **Deliverables**

You are required to turn in the following items in a zip file (**groupID\_Project2.zip**) in this assignment:

1) The source code of parallel PageRank you implemented.

2) The executable class file and the README file that describe its usage.

3) The output file (**groupID\_pagerank\_output.txt**) which contains the top 10 ranking url numbers.

Points will be reduced (maximum 0.5 points) if the filename or directory structure is different from instructed above.

# Evaluation

The total points for Project #2 is 10, where the distribution is as follows:

* 1. Completeness of your code and output (9 points)
  2. Readability and clarity of README.txt (1 points)

# Introduction

As a follow up of Project #1, you will implement a parallel version of PageRank by using MPI programming interface. We provide a pseudocode for MPI PageRank, and you will complete the implementation based on the pseudocode to make a working version. We provide functions for file IO so that you can focus on main implementation only. For more details on PageRank algorithm, you can refer to the previous descriptions for Project #1.

We expect you to implement the program using Java. There is a Java library called MPJExpress that provides the MPI functionalities. You should use this library for the implementation.

**Parallel PageRank**

Developing parallel PageRank is an active research area for both industry and academia, and numerous algorithms have been proposed. The key idea is to partition PageRank problems into N sub-problems so that N processes solve each sub-problem concurrently. One of the simplest approaches in partitioning is a vertex-centric approach: the graph of PageRank can be divided into groups of vertices and each group will be handled by a process. We take this approach for our MPI PageRank implementation.

# Pseudocode for MPI PageRank

You are given the MPIPageRank.java file with some utility functions. You are required to implement the missing functionalities. The ones you need to implement are clearly marked in the source code.

# Compile guide

Before compiling you need to download and set up MPJExpress in your machine. The instructions assume a Linux environment.

You can download MPJExpress from <http://mpj-express.org/>

You will receive a file called mpj-v0\_38.zip. Unzip this file and set the following environment variable.

export MPJ\_HOME=/path/to/mpj-unzipped-folder/

export PATH=$PATH:$MPJ\_HOME/bin

**Compile**

After the environment variables are set you can compile the Java program by using the following command.

Open up a terminal and cd to the homework directory. Then execute:

javac -cp .:$MPJ\_HOME/lib/mpj.jar MPIPageRank.java

This will produce MPIPageRank.class file, after which you are done with compilation.

**Execution**

**Single Machine**

To execute the program with MPI, use the following command in the homework directory:

mpjrun.sh -np 8 MPIPageRank pagerank.input.1000.0 output.txt .001 .85 30

-np means the number of processes. In the above case you are creating 10 processes and executing the algorithm in parallel, but this all happens in a single machine.

## Run MPJ with multi-nodes mode

If you’d like to run your program on a real cluster with distributed nodes, you can try this out as well.

Before you can run the program you need to install MPJExpress on the CS linux machine as well. You can follow the instructions above to do so.

Before running any MPJ job under CS Linux environment, you will have to make a “machines” file, which has the worker hostname written line by line, e.g. lh115linux-01.soic.indiana.edu and lh115linux-01.soic.indiana.edu. Note that the filename must be “machines.”

[skamburu@lh115linux-01 mpjmean]$ cat machines

lh115linux-01.soic.indiana.edu

lh115linux-02.soic.indiana.edu

Each group is assigned their own machines. Please use only those machines granted to you. The group assignments can be found in:

<https://docs.google.com/spreadsheet/ccc?key=0AtR8aHmmVF3ydG1uazA4Umk2M2poY2RfaWFpNlFEUEE&usp=sharing>

Once you have set up the “machines” file, you can use mpjboot to start MPJ daemons on worker nodes. Here, an IU passphrase may be required as you are starting the mpj daemons remotely.

skamburu@lh115linux-01.soic.indiana.edu's password:

Starting mpjd...

wrapper | Spawning intermediate process...

skamburu@lh115linux-02.soic.indiana.edu's password:

Starting mpjd...

wrapper | Spawning intermediate process...

[skamburu@lh115linux-01 lab\_mpimean]$

Then you can run MPJ PageRank by adding “-dev niodev” parameters to start your application across worker nodes.

mpjrun.sh -dev niodev -np 8 -MPIPageRank pagerank.input.1000.0 pagerank.out .001 .85 30

After you are done experimenting, execute the following command to clean up the resources.

mpjhalt.sh machines