Parallel and Distributed Programming Assignment 3

M veeramakali Vignesh, 2017CS10346

Kailash Kumawat, 2017CS10338

May 3, 2020

1 Introduction and Overview

We have worked on three different implementations of calculating pagerank of a set of web pages given their link structure using the technique of MapReduce. We used the existing libraries in part 1,3 and built our own MPI library in part 2.

2 Theoretical/Technical Background

2.1 About the problem

The page rank of a graph of pages can be calculated using power method as mentioned here.
Equations to compute:

$$I^{k+1} = \mathbf{H}I^k$$

$$\mathbf{G} = \alpha\mathbf{H} + \alpha\mathbf{A} + \frac{1-\alpha}{n}\mathbf{1}$$

$$\mathbf{G}I^{k+1} = \alpha\mathbf{H}I^k + \alpha\mathbf{A}I^k + \frac{1-\alpha}{n}\mathbf{1}I^k$$

2.2 MapReduce

MapReduce is an algorithm that composed of independent steps that can be parallelized. A MapReduce consists of 3 stages, namely **map**, **shuffle** and **reduce**Map stage consists of a number of independent map tasks, which filters, sorts and generates list key and value pairs per maptask. Shuffle stage consists of grouping up the values with the same key into a list for every key. In reduce stage, all the values with the same key are reduced to one key in a manner required by the problem. Finally we will have a list of unique key value pairs which would be the solution.

2.3 MapReduce for Pagerank

We need to perform the above mentioned multiplication successively until convergence to get the final pagerank scores. We can see that every step here is a matrix multiplication.

- Every column will be assigned to a maptask. The task will collect all the non zero values and multiply it with the corresponding element in the eigen vector. The key will be the row number and the value will be the corresponding product.
- The key value pairs will be shuffled and aggregated.
- The key-multivalues will then be reduced to a key-value by adding all the values in the list. this will be the value of the element with the index equal to key in the resultant vector.

For a general matrix this would take O(n2) time which is the same as normal matrix multiplication. But here each non-zero element in a column represents the links a page contains which is 10 on an average. So the whole process effectively becomes O(n).

3 Results And Analysis

3.1 Comparison across Parallel implementation

Table 1: Results (Time(s))

Benchmark	**mr-c++**	**my-mr-mpi**	**mr-mpi**	**Iterations**
erdos-10000	0.341	0.37	0.13	9
barabasi-20000	3.75	1.06	0.31	33
erdos-20000	0.6	0.47	0.17	9
barabasi-30000	5.95	1.55	0.36	35
erdos-30000	0.92	0.57	0.18	9
barabasi-40000	6.37	1.62	0.48	27
erdos-40000	1.28	0.65	0.27	9
barabasi-50000	8.87	2.34	0.59	31
erdos-50000	1.56	0.76	0.29	10
barabasi-60000	10.11	2.56	0.67	29
erdos-60000	2.08	0.92	0.35	10
barabasi-70000	13.59	3.32	0.77	34
erdos-70000	2.21	0.99	0.41	9
barabasi-80000	16.22	3.89	0.88	35
erdos-80000	2.54	1.14	0.43	9
barabasi-90000	17.18	4.09	0.93	33
erdos-90000	2.90	1.32	0.45	9
barabasi-100000	20.96	4.90	1.19	36
erdos-100000	3.20	1.36	0.47	9

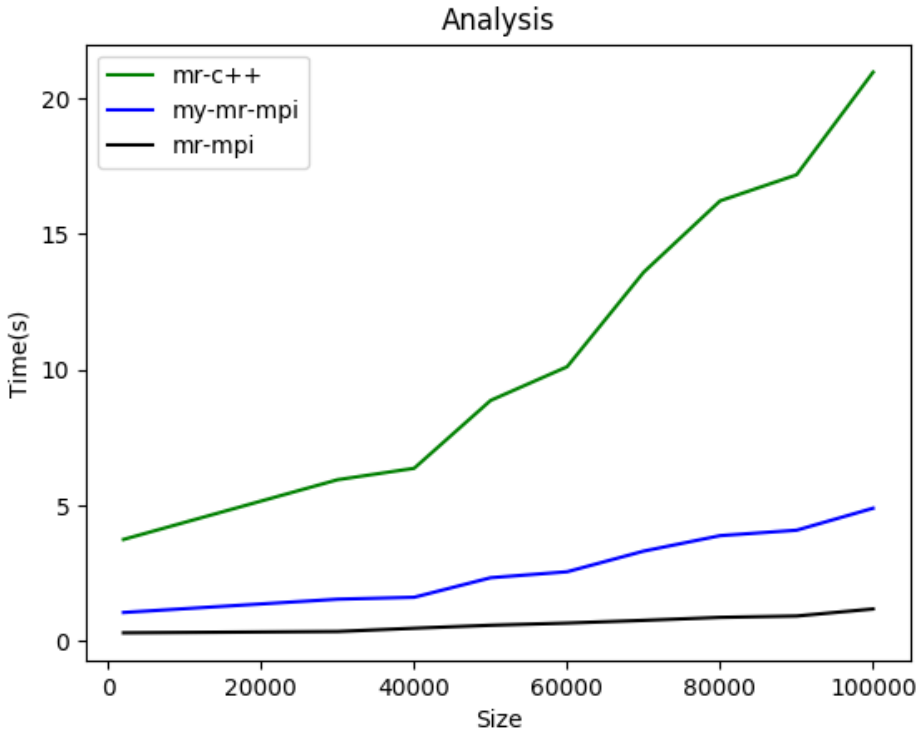

Figure 1: MPI_Recv code snippet

We can see that the pthreads implentation of mrc++ is the least efficient and mpi implementation is the fastest.