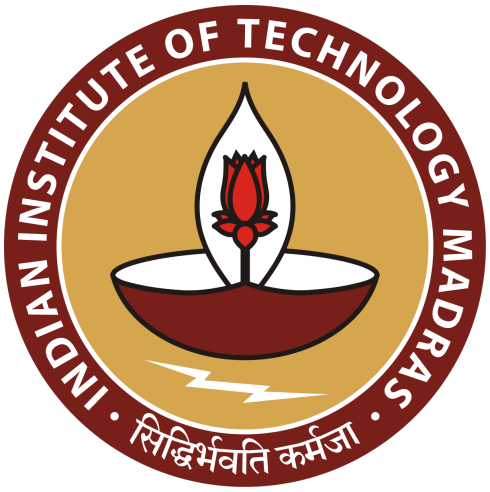
Indian Institute Of Technology Madras



ID5130 - Parallel Scientific Computing

Parallelizing Markov Chain Monte Carlo PageRank Algorithm

Ananthu Nair S. - AE17B109
Avinash G Bagali - AE17B110

30th May 2021

Abstract

Monte Carlo algorithms are a set of algorithms that perform repeated random sampling from a given set to model probability distributions that are not easy to predict directly due to random variables. A modification of these algorithms, called the Markov Chain Monte Carlo (MCMC) algorithm is a technique used for the evaluation of high dimensional probability distributions, which is not easy using simple Monte Carlo algorithms. However, it is seen to be an inherently sequential algorithm[1] due to its Markov property. The Metropolis-Hastings Algorithm is one of the most common MCMC algorithms. In this project, we propose to create a parallel implementation of a modified version[2] of MCMC algorithm, to solve the Page Rank problem in an information retrieval system.The Page Rank algorithm is used to assign a relevance score to a particular webpage based on the hyperlink structure that points to that particular page. It does this by creating a Markov chain transition matrix in which element (i,j) signifies the probability of a user to go to page j given that he is currently on the page i. The solution of the Page Rank would be the stationary state of the Markov chain.

Contents

1 Introduction **3**

2 Problem Modelling **3**

2.1 World Wide Web as Markov chain 3

2.2 Numerical solution using Monte Carlo 3

3 Theory **4**

3.1 Markov chain without dangling nodes 4

3.2 Markov chains with dangling nodes 4

3.3 Analytical solutions 5

4 Algorithms **5**

4.1 Parameters 5

4.2 Algorithm 1: Random Start 6

4.3 Algorithm 2: Cyclic Start 6

4.4 Algorithm 3: Cyclic Start based on the Whole Path 6

4.5 Algorithm 4: Cyclic Start on whole path with stopping at dangling nodes 7

4.6 Algorithm 5: Random Start on whole path with stopping at dangling nodes 7

5 Validation on small graph **7**

5.1 Observations 7

5.2 Inferences on validation 9

6 Parallelizing using OpenMP **9**

7 Real World Graph **9**

7.1 Introduction 9

7.2 Parallel code validation with Algorithm 5 10

7.3 Comparison of Algorithms on the Real World Data 11

7.4 Evaluating Parallel Performance 12

1 Introduction

The internet is one of the greatest creation of mankind containing vast amounts information from various domains and ages. None of this information would be of any use to us without a Search Engine. A Search Engine consists of different preprocessing steps that allows it return millions of results at the click of a button. One of the preprocessing steps is to "crawl" through the web and rank the relevance of different webpages. Once a web crawler has retrieved information about different webpages and hyperlinks, we can use the page rank algorithm[3] to rank these webpage in the order of their importance. The idea is that pages linked from an "important" page are also important. This creates a problem of circularity similar to solving a set of linear equations, where we want to find the importance value associated with each webpage.

2 Problem Modelling

2.1 World Wide Web as Markov chain

The World Wide Web can be represented as a directed graph where the nodes are the webpages and the directed edges between the nodes are the hyperlinks between the pages. Thus the surfing of the web can be modelled as a Markov chain process and the importance of a page can be gauged using the probability a random surfer on the internet will land on that particular page. The importance vector is thus the stationary state vector of the Markov chain associated with surfing the World Wide Web.
At the stationary condition, we have

$$\pi Q = \pi$$

where,
π is a row state vector,
Q is the state transition matrix
thus π is the eigenvector associated with the eigenvalue of 1 for the matrix Q.

2.2 Numerical solution using Monte Carlo

The process of solving the Markov chain problem analytically involves the inverting of a (NxN) matrix, where N are the number of nodes in the graph. This will become computationally infeasible as the number of nodes increases, especially in the case of the WWW which has billions of web pages. To overcome this, we use a numerical approximation to the problem using Monte Carlo methods, in which the surfer starts at

a node and surfs the web by choosing one of the outgoing edges randomly at each step till it hits a stopping condition. We need to take care of some edge cases to make sure we get a correct result, (for e.g. how do we take care of the dangling nodes (nodes with no outgoing links). We go over this in detail in the next section.

3 Theory

3.1 Markov chain without dangling nodes

The typical state transition matrix for a Markov chain model Q is given as

$$Q_{i,j} = \begin{cases} 1/N_{neigh_i} & if - page - i - links - to - page - j \\ 0 & otherwise \end{cases}$$

where N_neigh_i are the number of outgoing links from i_{th} node.

3.2 Markov chains with dangling nodes

However, a web with dangling nodes (pages with no outgoing links) produces a matrix Q which contains one or more rows of all zeroes, Such a matrix must have all eigenvalues less than or equal to 1, but 1 need not be an eigenvalue for Q and thus, we may not have a stationary state vector. To be able to use some properties of we make a modified transition matrix P given as

$$P_{i,j} = \begin{cases} 1/N_{neigh_i} & if - page - i - links - to - page - j \\ 1/N & if - page - i - has - no - outgoing - links \\ 0 & otherwise \end{cases}$$

where,
N_{neighi} are the number of outgoing links from i_{th} node. N is the total number of nodes in the graph.

P can be thought of as the transition matrix Q along with the property that when we encounter a dangling node, we randomly jump to a new node.

In our random walk, if we reach a dangling node, we can either stop the walk at that point or continue by jumping to a random point. The analytical solutions in each case are given as

3.3 Analytical solutions

If we stop at dangling nodes

If we stop upon reaching a dangling node, the analytical solution is in this case is given by:-

$$\pi_j = \gamma \sum_{i=1}^{n} [I - cQ]^{-1}_{ij}$$

where, $\gamma = \sum_{j=1}^{n} \sum_{j=1}^{n} [I - cQ]^{-1}_{ij}$

If we continue

If we continue by randomly choosing one of the N nodes, the analytical solution is in this case is given by:-

$$\pi_j = \frac{(1-c) * \sum_{i=1}^{n} [I - cP]^{-1}_{ij}}{n}$$

for more details on the derivation, refer to [2]

4 Algorithms

A typical Plain Random walk algorithm is as follows:

1. Start at Initial Node in the graph.
2. Randomly choose one of the outgoing edges from that node and travel along that edge to the next node.
3. Repeat Steps 1 and 2 till a stopping conditions is met.

Our Monte Carlo method can be altered using the following parameters. These parameters allow us to tune our model to take care of certain situations. for e.g. at each page a surfer may/may not click on the next page or what do we do if we reach a page which has no outgoing links.

4.1 Parameters

- **C (damping factor)** :- probablity surfer continues to click on next page (set to 0.85 throughout)
- **Initialization** - We can choose the starting Node by using either of these methods.
  - Random Initialization

  - Cyclic initialization :- m number of walks start from each of the N nodes. (total m*N walks)

- **Stop @ Dangling Nodes** :- When we encounter a node that doesn't have any outgoing links, we can do one of the following
  - Stop the walk at that particular Node.
  - Continue the walk by randomly jumping to one of the N Nodes.
- **Importance of page** :- Importance of the i^{th} can be evaluated using
  - The fraction of Walks that **ends** on the i^{th} Node.
  - The fraction of times the i^{th} Node was **visited** cumulatively across all the walks.

We get a family of algorithms by changing changing the above given parameters and flags. We will introduce a few of them in the next section.

4.2 Algorithm 1: Random Start

C = 0.85 (probablity that surfer continues clicking)
Initialization = Random Init
Stop @ Dangling Nodes = Continue by randomly jumping to one of the N Nodes.
. **Importance of Page** = fraction of Walks that end at i^{th} Node

4.3 Algorithm 2: Cyclic Start

C = 0.85 (probablity that surfer continues clicking)
Initialization = Cyclic Init, m walks start from each of the N Nodes
Stop @ Dangling Nodes = Continue by randomly jumping to one of the N Nodes.
. **Importance of Page** = fraction of Walks that end at i^{th} Node

4.4 Algorithm 3: Cyclic Start based on the Whole Path

C = 0.85 (probablity that surfer continues clicking)
Initialization = Cyclic Init, m walks start from each of the N Nodes
Stop @ Dangling Nodes = Continue by randomly jumping to one of the N Nodes.
Importance of Page = fraction of times the i^{th} Node was **visited** cumulatively across all the walks.

4.5 Algorithm 4: Cyclic Start on whole path with stopping at dangling nodes

C = 0.85 (probablity that surfer continues clicking)
Initialization = Random Init
Stop @ Dangling Nodes = Stop the walk at a dangling node
Importance of Page = fraction of times the i^{th} Node was **visited** cumulatively across all the walks.

4.6 Algorithm 5: Random Start on whole path with stopping at dangling nodes

C = 0.85 (probablity that surfer continues clicking)
Initialization = Random Init
Stop @ Dangling Nodes = Stop the walk at a dangling node
textbfImportance of Page = fraction of times the i^{th} Node was **visited** cumulatively across all the walks.

5 Validation on small graph

We first run a serial MCMC algorithm and validate it against the analytical solution for a small graph consisting of 4 nodes and 6 edges as shown below.

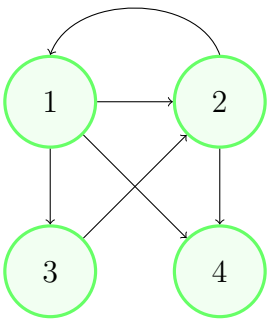


Figure 1: The validation graph used. The green circles represent the nodes.

We perform the validation using each of the algorithms mentioned above on the small graph. We then compare it with the results obtained from the analytical solution. These should roughly match to ensure that that code is running correctly.

5.1 Observations

For the case of algorithms 1, 2 and 3, the analytical solutions are the same since there are no changes to the main algorithm in these. Only the initialization and the value used

for the page rank estimation change. For the algorithms 4 ad 5, the main part of the algorithm changes since we deal with the dangling nodes differently. This also changes the analytical expression, since the matrix changes now. Now we compare the results of this analytical solution with all the solutions obtained from these algorithms.

Algorithm	**Page Rank**	
	Average MCMC	Analytical
1	[0.2450, 0.1900, 0.2700, 0.2950]	[0.2309, 0.1659, 0.3069, 0.2963]
2	[0.2525, 0.1425, 0.3050, 0.3000]	[0.2309, 0.1659, 0.3069, 0.2963]
3	[0.2351, 0.1594, 0.3150, 0.2891]	[0.2309, 0.1659, 0.3069, 0.2963]
4	[0.1816, 0.0942, 0.2492, 0.4749]	[0.2309, 0.1659, 0.3069, 0.2963]
5	[0.1668, 0.0925, 0.2623, 0.4784]	[0.2309, 0.1659, 0.3069, 0.2963]

Table 1: The page ranks obtained from the Markov Chain Monte Carlo for various algorithms and the analytical solution.

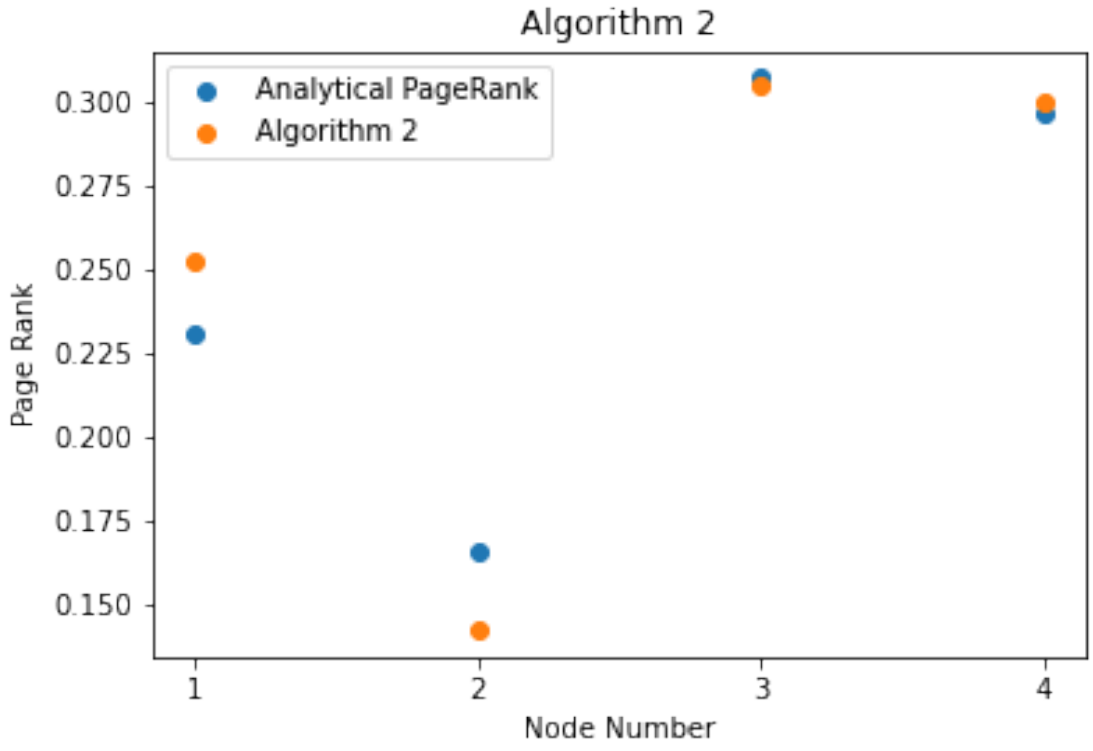


Figure 2: Plot showing the page rank as obtained analytically and using the algorithm 2.

From the plot it can be seen that the trends are followed as expected.

5.2 Inferences on validation

- For MCMC, the solution is not the same value always since there a random factor involved in making the choice of the path followed. Hence it is averaged to have a better metric to compare with the analytical page rank.
- The analytical Page rank and the numerical page rank shows a similar trend in their values, giving a higher priority to the nodes with the higher number of inlinks.
- When the dangling nodes terminate the walk on encounter, it can be seen that the page rank gets skewed towards these dangling nodes. This is to be expected, since more pages stop at these and does not move forward.

6 Parallelizing using OpenMP

An advanntage of this random walk algorithms is that they have an inherent parallelizability, since each of the random walks are independent of each other. Therefore, multiple walks can be performed simultaneously without any loop dependence. Hence the parallelization of these algorithms are easy to implement. In the code, the for loop which iterates over each walk can be parallelized using the following pragma.

```
1    #pragma omp parallel for num_threads(thread_count)
```

However, there is a need for reduction since multiple threads might try to update the Page rank of the same page. Hence the final modified pragma for the code is as shown.

```
1    #pragma omp parallel for num_threads(thread_count) reduction(+:
   PageRanks)
```

This is then used on the small graph and similar results as the serial case was obtained. Thereby proving the correctness of the parallelization.

7 Real World Graph

7.1 Introduction

The real world graphs showing a webpage network is huge and might contain millions of nodes. Hence storing them in matrices and using the analytical solution for Page Rank becomes infeasible as mentioned earlier. For this analysis, we use a network graph that is openly available, made available by the Stanford University, as a part of the Stanford Netword Analysis Project[4]. Here only the graph provided is being used even though the project makes available many other tools for graph manipulations. The specific graph used for this analysis is the Stanford Web Graph from the same source. The graph contains 281903 nodes and 2312497 edges.

7.2 Parallel code validation with Algorithm 5

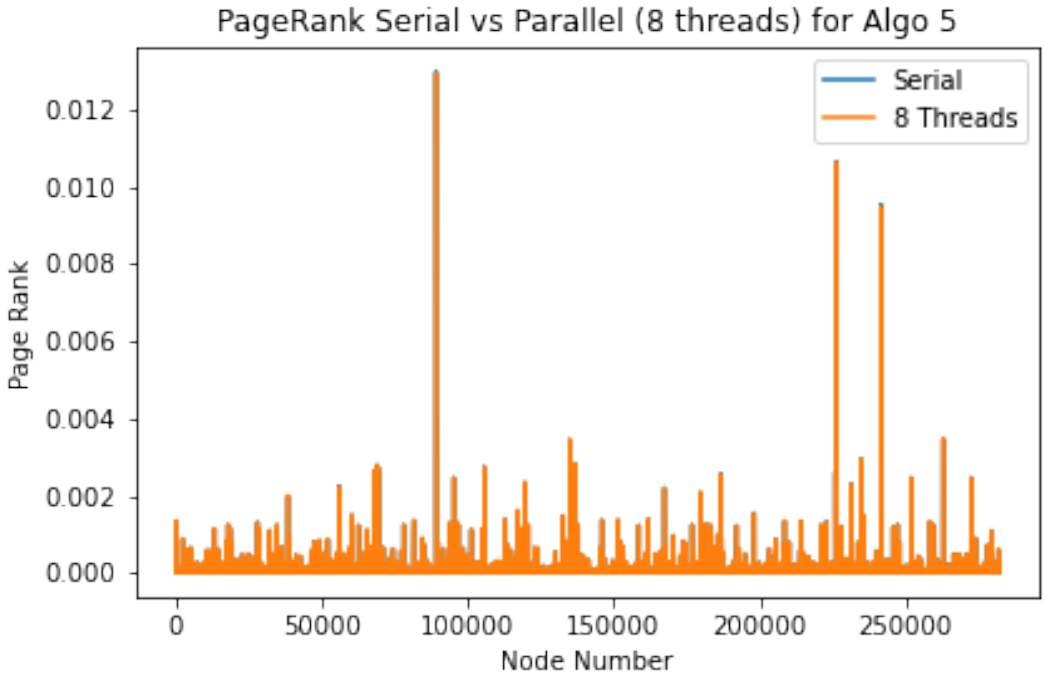
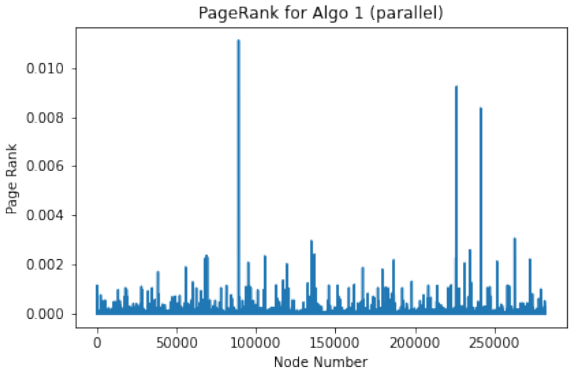


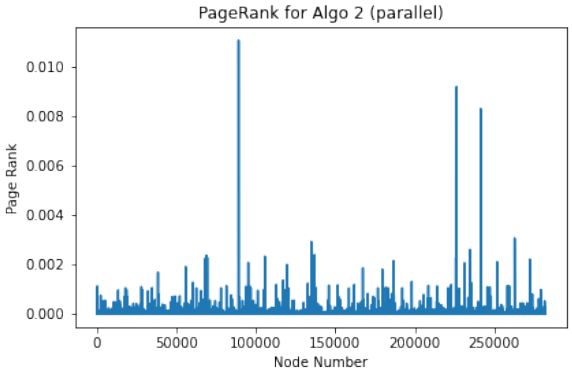
Figure 3: Plot showing the page rank as obtained using the algorithm 5 serially and parallelly run with 8 threads

From the plot it can be seen that the page rank solutions match exactly and overlaps for the serial and the parallel versions of the algorithm 5. This proves that the parallel code works as expected.

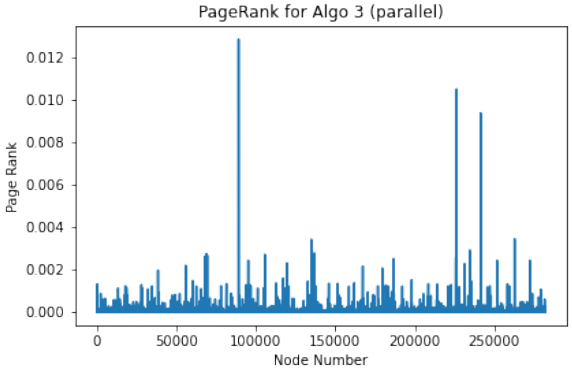
7.3 Comparison of Algorithms on the Real World Data



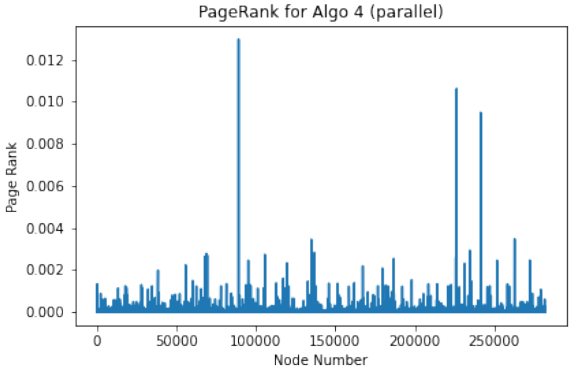
(a) Plot showing the page rank as obtained using the algorithm 1



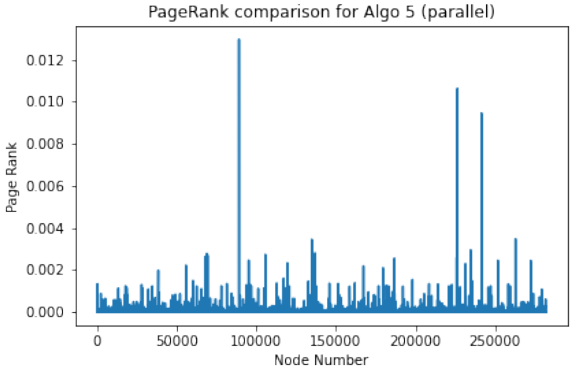
(b) Plot showing the page rank as obtained using the algorithm 2



(c) Plot showing the page rank as obtained using the algorithm 3



(d) Plot showing the page rank as obtained using the algorithm 4



(e) Plot showing the page rank as obtained using the algorithm 5

Figure 4: Plots to compare the Page rank obtained from Algorithm 1, 2, 3, 4, and 5. These were all run with 8 threads.

The above plots show the Page rank obtained from each of the algorithms. They follow the similar trend throughout, albeit with minor differences among each of them.
It can be especially seen from the above graph that there is a slight difference in page ranks obtained using the two algorithms 1 and 5. In algorithm 5, we terminate the walk

when a dangling node is reached, whereas in algorithm 1, it continues by jumping to a randomly chosen node. Due to this, the algorithm 5 gives a higher weight to the dangling nodes as expected, whereas the algorithm 1 does not. Similarly, the algorithm 4 also gives a higher rank to the dangling nodes.

7.4 Evaluating Parallel Performance

We now perform the random walk serially and parallelly and compare the time taken for each case. The parallel code is run using 2, 4 and 8 threads.

First set of values are obtained using $m = 50$. Hence the total number of walks is mN_{Nodes}, i.e, 14,095,150 random walks.

Number of Threads	**Time Taken** (in s)	**Speedup**
Serial	34.7506	-
2	19.8518	1.7505
4	12.5470	2.7696
8	8.5522	4.0633
16	6.7755	5.1289

Table 2: The time taken and the speedup obtained from running the parallel version of the code for 2, 4, 8 and 16 threads for $m = 50$.

Now, we do the same for $m = 100$. Hence the total number of walks in this case 28,190,300.

Number of Threads	**Time Taken** (in s)	**Speedup**
Serial	73.5711	-
2	38.8241	1.8949
4	23.9799	3.0680
8	16.7704	4.3869
16	13.4376	5.4750

Table 3: The time taken and the speedup obtained from running the parallel version of the code for 2, 4, 8 and 16 threads for $m = 100$.

So as it can be seen, when the number of processes are increased, the time taken for the walks reduces drastically. However, as the number of processes are increased, the value of speedup obtained reduces. Moreover, for a higher number of walks (higher value of m), the speedup obtained is higher. Both these observations are expected as based on Amdahl's effect.

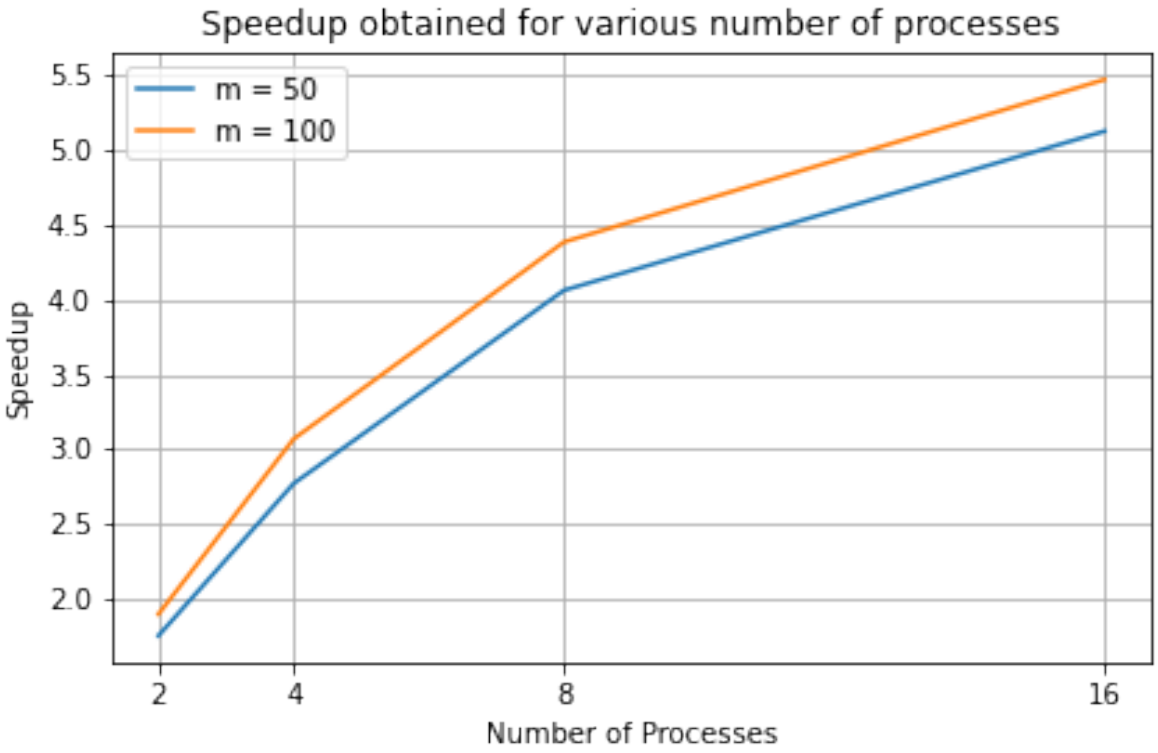


Figure 5: Plot showing the speedups obtained for various number of processes used with values of m 50 and 100.

References

[1] B. Calderhead, "A general construction for parallelizing metropolis-hastings algorithms," *Proceedings of the National Academy of Sciences*, vol. 111, no. 49, pp. 17408–17413, 2014.

[2] Ö. Salehi, "Pagerank algorithm and monte carlo methods in pagerank computation," 2011.

[3] Wikipedia contributors, "Pagerank — Wikipedia, the free encyclopedia." `https://en.wikipedia.org/w/index.php?title=PageRank&oldid=1024516955`, 2021. [Online; accessed 30-May-2021].

[4] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection." `http://snap.stanford.edu/data`, June 2014.