B. Tech Program Third Year

Course: Parallel Programming

Course Code: DS3202

## “implement PARALLEL CODE in google pagE rank algorithm ”

by

(Reg. No: )

(Reg. No:)

Under the guidance

of

**Mr.**

**Assistant Professor**

Information Technology | SIT | Manipal University Jaipur, Jaipur (Raj.).

**Department of Information Technology**

**School of Computing and Information Technology**

**Faculty of Engineering**

**Manipal University Jaipur, India**

April, 2023

# Certificate

This is to certify that the project titled “**IMPLEMENT PARALLEL CODE IN GOOGLE PAGE RANK ALGORITHM**”is a record of the bonafide work done by submitted for the partial fulfilment of the requirements for the completion of the Parallel Programming (DS3202) course in the Department of Information Technology of Manipal University Jaipur, during the academic session Jan - May 2023.

|  |
| --- |
| *Signature of the mentor* |
| Mr.  Assistant professor IT  Department of IT |
| *Signature of the HoD* |
| Head of the Department  Department o |

# PROBLEM STATEMENT

# The PageRank algorithm is a widely used algorithm for ranking web pages based on their importance and relevance. However, as the size of the web grows, the computational cost of computing PageRank scores for all pages in the graph becomes increasingly problematic. To solve this issue, parallel computing techniques can be used to distribute the computation across multiple processors or nodes.

# The problem addressed in this project is to implement parallel code in the Google PageRank algorithm, with the goal of accelerating the computation of PageRank scores for large web graphs. The parallel code is designed to parallelism algorithm, such as the independent computation of PageRank scores for each node.

# The project will involve a analysis of the sequential PageRank algorithm and its computational requirements, as well as the design and implementation of parallel algorithms using parallel programming OpenMP with shared memory. The performance of the parallel algorithms will be evaluated using benchmarks and metrics such as speedup, efficiency, and granularity, and the results will be compared to those of the sequential algorithm. The ultimate goal of the project is to provide a scalable and efficient implementation of the PageRank algorithm that can handle large web graphs in a reasonable amount of time.

# IMPLEMENTATION OF PAGE RANK

# SEQUENTIAL IMPLEMENTATION

# In the first step of the project, we are tasked with the sequential implementation of PageRank. Once the implementation is completed, we need to track its execution times for several input sets, which are freely available at the Stanford Large Network Dataset Collection.

# The algorithm we use to implement PageRank is the Power Iteration Method. The power iteration method is a simple and widely used algorithm for computing the dominant eigenvector and eigenvalue of a square matrix. In the context of the PageRank algorithm, the power iteration method is used to compute the PageRank scores for each node in the graph. The basic idea of the power iteration method is to repeatedly apply the matrix to a vector, and normalize the resulting vector after each iteration. By computing the PageRank vector using the power iteration method, the PageRank algorithm is able to assign a score to each node in the graph based on its importance and relevance.

# In the real web context, to perform the PageRank analysis correctly we need to check that the adjacency matrix A is stochastic, irreducible and aperiodic. To obtain a matrix that satisfies the first condition we need to add a complete set of outgoing links from each page with no out-links (the dangling pages described before) to all the pages on the Web. In our code this is done right before the beginning of the power iteration method. To obtain a matrix coherent with the second and third properties, instead, we need to add a link from each page to every page and give each link a small transition probability controlled by a parameter d. Such parameter is called the damping factor, which is set to 0.85 as per the PageRank paper.

# 

# PARALLEL IMPLEMENTATION-

# We parallelize the sequential implementation using SIMS SSE, shared memory. Once this is completed, we need to:

# • measure speedup and efficiency as a function of the processors and cores exploited by the implementation for a couple of different data sets.

# • measure how the execution time changes when the problem size is increased, without changing the number of processors and cores employed. This is done by considering different data sets, with different amounts of data.

# We implement the parallelization n using a shared memory system with POSIX threads as follows:

# #pragma omp parallel for schedule(static) if(parallel)

# num\_threads(numthreads)

# 

# Notice we can insert in the beginning of the code the number of threads and the desired granularity. In the tests, we have experimented with several combinations of threads (4, 8, 16) and granularities to obtain the best possible results; we have also tried changing the flag from static to dynamic in order to test more combinations of the execution.

# System configuration –

# Intel Core i7-8550U (8th Gen)

# 1.8GHZ

# RESULTS

# Dataset magnitutdes:

# 

# Our implementation is written in C and uses POSIX threads to parallelize the PageRank algorithm. The actual parallelization is performed over a for loop that must be computed in parallel. Experimenting with the number of threads we have noticed that, as expected, using less threads than the number of cores of the CPU is not useful, since some cores remain idle. Using too many threads, instead, increases the communication cost of the scheduling. As for the granularity of the decomposition, we have performed several tests to get the best possible value for each data set. In fact, only by assigning a correct portion of computation to each thread the parallel computation can improve its performance. Studying the results, we can observe that the speedup is not extremely high: the computational times are acceptable even when using the sequential version of PageRank.

# 

# On further implementation on other devices, (system configurations of – Intel Core i5-10210U 2.1GHz) we secured other timings for serial and parallel code for the Stanford dataset –

# With numthreads-16

# Scheduling – dynamic

# Granularity – 1/8

# Serial time – 0.404114seconds

# Parallel time – 0.382979seconds

# Speedup – 1.05518

# Efficiency – 0.069

# As we can see the results are overall better, with smaller execution times. We

# include some plots to visually describe the results of our tests:

# 

# CONCLUSION

# In conclusion, the parallel implementation of the PageRank algorithm provides significant speedup over its serial counterpart. By leveraging the power of multi-core processors, the algorithm can be parallelized to take advantage of the available resources, resulting in faster execution times. In our project, we implemented the PageRank algorithm using OpenMP and tested it on a dataset with approx. 281903 web pages and 2312479 edges(links).

# Moreover, we analyzed the performance of the parallel algorithm by varying the number of threads and the granularity of the computation. Our experiments showed that the optimal number of threads and granularity depend on the characteristics of the dataset and the hardware configuration. We also evaluated the scalability of the algorithm by increasing the size of the dataset and showed that the parallel algorithm scales well with the size of the input. The results of our experiments can be used to optimize the parallel implementation for different datasets and hardware configurations, and to further improve the performance of the algorithm in practice.

# REFERENCES

# Figueredo and Wolf, 2009] Figueredo, A. J. and Wolf, P. S. A. (2009). Assortative pairing and life history strategy - a cross-cultural study. Human Nature, 20:317–330.

# [S. Brin and L. Page, 1998] . S. Brin and L. Page (1998). The anatomy of a largescale hypertextual web search engine. Computer Networks and ISDN Systems,33:107-117