Counting Mutal Web Linkages

Candidate 15113

April 2020

1 Abstract

In this report we studied some of the important algorithms for studying and analysing web-graphs. Parallelization was used, and a significant speedup was gained. We also found the most efficient algorithms for counting web linkages out of the ones we used.

Contents

2 Introduction

For this study we were asked to make a program that can count the total number of web linkages for a given graph, as well as finding the sites with most occurrences. Our code was written in C programming language and was parallelized using the OpenMP library. We used two different methods for storing the data given from the web-graphs. The methods were used were ordinary sparse matrix as well as the CRS format[1]. Besides that, benchmark for the algorithms and test functions are included.

3 Method

3.1 Web-graph

The web-graphs we have used have all the same structure. However, the problem with them is that in their current format, they are not in a format a machine could process. Therefore, the data has to be stored in arrays. We used two different arrays to store our data. The methods are explained bellow.

3.2 Dense Matrix

Implementing the dense matrix for this kind of problem is not so difficult. We start with a matrix $M_{i,j}$ on the form

$$\mathrm{M}_{i,j} = \begin{cases} 1 & j \text{ links to } i \\ 0 & \text{Else} \end{cases}$$

The dense matrix a symmetrical matrix where the number of rows/columns corresponds to the number of web-pages(i.e a $5x5$ matrix will have 5 web-pages). The indices i, j corresponds to two web-pages. If the web-pages are linked together, the indices in the matrix $M_{i,j} = 1$, else if they have no link, the indices in the matrix $M_{i,j} = 0$

3.3 CRS-Format

The second algorithm we used to store our data was the Compressed Row Storage or CRS-Format. When implementing the CRS-Format, we start by storing our data in two different arrays. All the the column indices will be stored in an array of size equal to the total number of links. We also need to store information about which row element the indices above belong to. To do this, another array needs to be allocated with size $N+1$ (N is the number of web-pages) with elements that represents when the indices above moves to the next row.

What we do here is that we have an array with the number of links per row, which will be used to sum over all rows which will give us the the row pointer we wish.

3.4 Counting links using Dense Matrix

Using our dense matrix representation, we can compute the total number of web linkages by iterating through every element in the matrix. When iterating, we look for the values where $M_{i,j} = 1$ and store them by row in using a counter. After that, we go through every row and count the times we get the value 1. When this is done, adding the element number for our counter to the array which holds the number of involvements will give us. This is done for every row. When going over all rows, we will get the amount of times a single element was involved.

3.5 Counting links using CRS-Format

Counting the mutual web linkages for CRS-Format is quite elegant. We iterate through the rows. Then we extract the number of elements in each row. That will give us the number of involvements for each column index if we take the number of row elements then subtract one.

3.6 Finding the top webpages

Since we have the necessary data stored in an *num_involvements* array, finding the top web-pages, is straight forward. The only thing we need to do is finding the elements with the highest values to lowest. The array is then printed out together with another array containing the web-page numbers. This allows us to see which web-pages has the top results.

3.7 Parallelization

In this project, the two functions which finds the total number of web links were parallelized using OpenMP.
Paralleizing in this case is simple. We just need to add the *pragma omp* decorator before our first for-loop in both our functions. Then we specify in our decorators that we are not going to use false sharing

4 Results

The results were tested on an Intel i5-4300U 1.90GHz with 4GB of RAM. I had access to a total of two cores with one thread each. Meaning total 2 threads. The tests were ran on Ubuntu 18.04

4.1 Benchmark

All the tests were done without compiler flags.

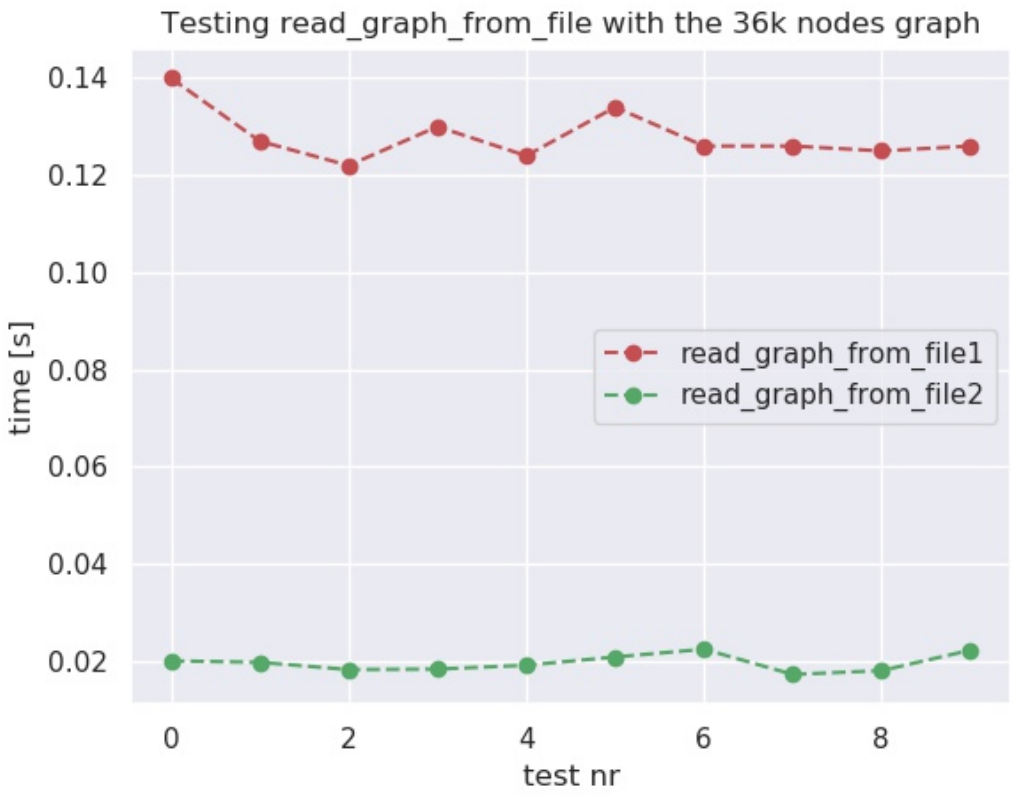

Figure 1: Tests the speed of our two files that reads the data. As we can see, the one where we implement the CRS-Format is more efficient

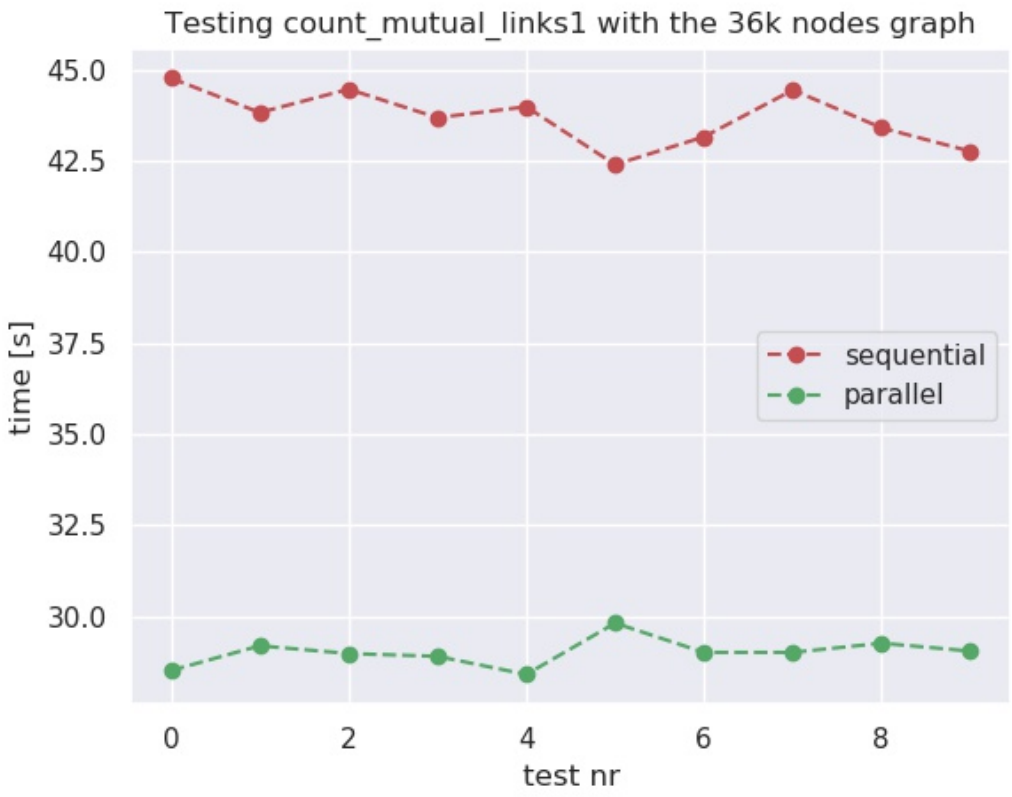

Figure 2: Here we counted mutual links using the data stored in the dense matrix. What we can see is that parallelizing gives a significant speedup. This speedup can be increased even more with more threads

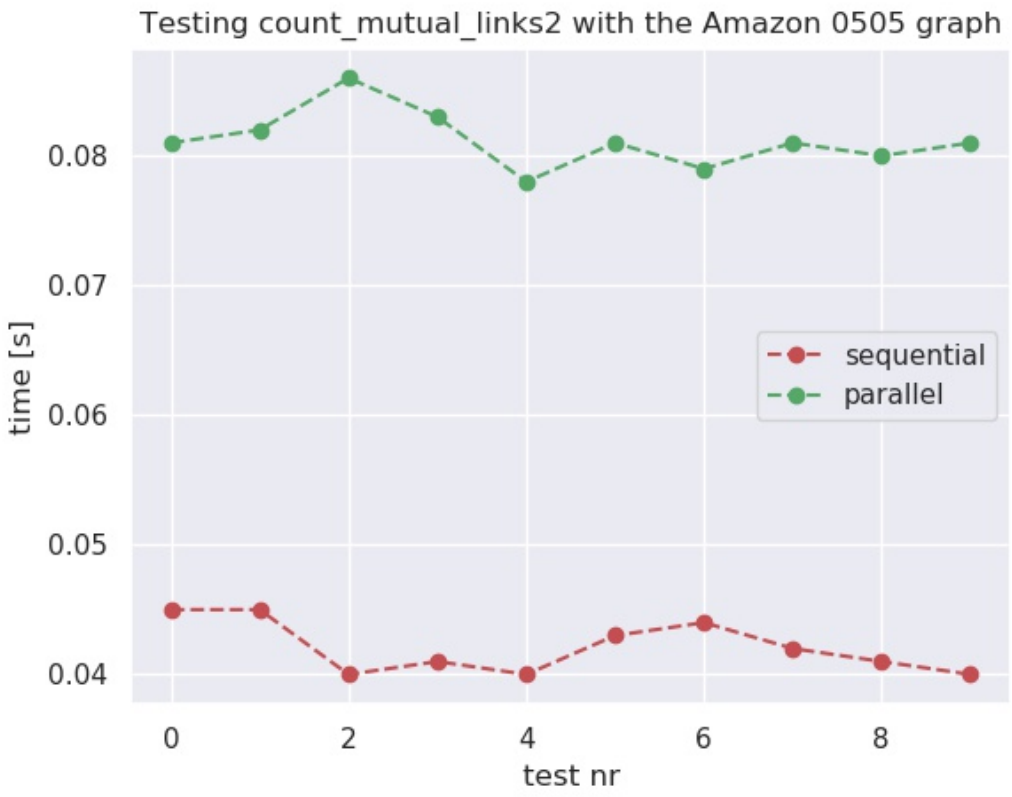

Figure 3: Here we counted mutual links using the data stored in the CRS-Format. This program was running so efficient that I had to use the largest data-set I could find (Amazon5050).

5 Conclusion

The first thing noticed was that the CRS-Format is way more efficient when it comes to computation. This can be seen by comparing Table 2 and Table 3.The dense matrix method did not work with large files like the *web-NotreDame.txt* file, because of memory limitations, therefore, for big files, we had to use the CRS-Format method.
Besides that, parallelizing made a significant difference in computation speed when using the dense matrix implementation. The computation could have been improved even more if I would have had access to more threads. Moreover, parallelization did not decrease the time for computation when CRS-Format was used. This is likely because the computations were so fast that OpenMP made

no difference.
In conclusion, the CRS-Format was shown to be superior in therms of computation speed compared to the dense matrix. But, in terms of parallelization, the files were too small for OpenMP to make a difference. Therefore, I cannot conclude that a speedup will be gained by parallelizing the CRS-Format function, but the possibility it will is likely.

References

[1] Georg H Gethard W. *Introduction to High Performance Computing for Scientists and Engineers*, pages 86–89. CRC Press, 2011.