IN4200 - Home Exam 1: Counting mutual web linkage occurrences

Kristoffer Langstad
Candidate number: 15227
krilangs@uio.no

Abstract

In this report we first describe the most important algorithmic and programming info done in C. Then we look at the time measurements of the implemented functions we have used, both with and without parallelization.

1 Data

The coding is computed with a Windows 10 64-bit operating system. It has an Intel(R) Core(TM) i7-7700HQ CPU with 4 physical cores and 8 virtual cores. The codes (and Makefile) should also work on a Linux system as explained in the **README.txt** file.

Then example-data of real-world web graph files are downloaded from `https://snap.stanford.edu/data/web-NotreDame.html`. The file format of the web graphs we look at are further explained in the project description [1]. We have used three different text files:

1. **8-webpages.txt**: This is the project example that is converted to a web graph file containing the linkage connection information. This example is used to confirm that our functions give the same/correct values as the example does.

2. **p2p-Gnutella05.txt**: This file is used mostly for the 2D table format for reading the web graph info, since this text file has the biggest number of nodes and edges the 2D table can allocate in memory (at least by our implementation and tested sizes of various text files for this computer system).

3. **web-NotreDame.txt**: This is a large real-world web graph file used for the CRS format for reading the web graph info.

2 Code structure

All the main code is done in C, and are contained in the zip file **IN4200_HE1_15227**. This zip file also contains the text files needed, a **README.txt** file for compilation explanation, a Makefile and we have also included the Python 3.7 file for plotting the time measurement of the parallelization.

Each function is implemented into its own separate *.c file with the same filename as the function name. The names of the functions (*) are:

- read_graph_from_file1
- read_graph_from_file2
- count_mutual_links1 & count_mutual_links1_OMP
- count_mutual_links2 & count_mutual_links2_OMP
- top_n_webpages & top_n_webpages_OMP

These functions are imported by other programs which do the main computations by making use of these functions. Each function is then timed, and parallelized with OpenMP where we separate the non-parallelized and parallelized functions run by separate test programs.

3 Code description

3.1 Functions

3.1.1 read_graph_from_file1

This function opens a given web graph text file, with the file format as explained in the project description, allocates a 2D table of zeros of dimension $N \times N$ with N number of web pages, and reads the web graph info into the table. 1's in the table are given when there is a direct link from a web page outbound to a web page inbound, excluding self-links. Here columns are noted as outbound and rows are noted as inbound. The function skips unwanted lines in the files with text at the top, only extracting the number of web pages N (nodes) and direct linkages N_{links} (edges), and then check for direct linkages for the rest of the file to add to the table. This 2D table is later used for counting mutual linkages.

3.1.2 read_graph_from_file2

This function does the same as first function, only now with a compressed row storage (CRS) format instead of a 2D table format. This is much more memory efficient since we will often have a lot of unnecessary 0's in the 2D table, and we only care about the 1's.

This format uses two 1D arrays of integer values. One **col_idx** array of length N_{links} storing the column indices corresponding to the direct links row by row, and one **row_ptr** array of length $N + 1$ containing index values at which new rows start where there are direct links. These two arrays are later used to count the mutual linkages.

3.1.3 count_mutual_links1

This function uses the 2D table made in the **read_graph_from_file1** function with 0's and 1's to count the total number of mutual web page linkage occurrences as well as the number of involvements per web page outbound for such mutual linkage occurrences. It stores the number of involvements per web page as outbound in an already allocated array **num_involvements** of length N (number of web pages). So when an element on a column in the 2D table is 1, we run through the rest of the rows on that column and add one for each mutual linkage (each 1 that occurs), and subtract by one to exclude the node itself. This is done for all rows.

The occurrence for a mutual web linkage happens when two web pages i and j as outbound are both directly linked to a third web page k as inbound. The total number of mutual occurrences is the sum of the number of involvements for all web pages or the sum of the indices in the **num_involvements** array divided by two, since we have to account for double counting of the linkages.

This function is parallelized with OpenMP in a separate function with the same name ending in _OMP, and used in another program utilizing parallelization.

3.1.4 count_mutual_links2

This function uses the CRS format made in the **read_graph_from_file2** function with the two 1D arrays to count the total number of mutual web page linkage occurrences as well as the number of involvements per web page outbound for such mutual linkage occurrences, like the **count_mutual_links1** function.

With the **row_ptr** array, we take the difference $row\_ptr[i+1] - row\_ptr[i]$ to get the number of index values in **col_idx** between the beginning of two rows. For the length of this difference when bigger than 1 (if equal to one then no mutual linkages on that row) we use the $row\_ptr[i]$ as the index of **col_idx**, and then this as the index of **num_involvements** where we add the difference above minus one (minus one accounting for the current node) for each number of web pages.

The total number of mutual occurrences is the sum of the difference above minus one for each number of web pages and then divided by two, like in the previous case to account for double counting of linkages.

This function is also parallelized with OpenMP in a separate function with the same name ending in _OMP to be used in the program utilizing parallelization.

3.1.5 top_n_webpages

This function is used to find the n chosen top web pages with respect to the number of involvements in mutual linkages, and prints these web pages and their respective numbers of involvements. To find the top involved web page, we find the largest value of the array **num_involvements** by comparing the values of the indices in the array. The largest value and its corresponding index is stored, and the largest value is then set to zero. The largest value in the array is then the second most involved web page. This process is repeated until we have found the chosen number n top web pages and their respective number of linkages.

This method may be altered if we were to use the **num_involvements** array further, since we here changes the values in the array when we have found the wanted largest value of the array. For this project, this change is not necessary to account for.

This function is also parallelized with OpenMP in a separate function with the same name ending in _OMP to be used in the program utilizing parallelization.

3.2 Test programs

These functions are used in three separate test programs (*.c) for reading the web graph info and counting mutual web linkages with 2D table and CRS format, and for finding the top web pages regarding involvements in mutual linkages with the CRS format.

3.2.1 Test_example

This is the test program of the project example with the **8-webpages.txt** file, which tests the correctness of the functions we have implemented to the project example where we know and print the results for comparison. This program just runs and prints out the wanted results made by the functions without any time measurements. The known values of this project example are added manually in the program to be printed as well for easy comparison between the results.

3.2.2 Test_functions

This program runs the computations of the functions with both a 2D table and CRS format of **p2p-Gnutella05.txt** and **web-NotreDame.txt** files. The program runs and

prints the wanted results and does a time measurement of each of the functions as well. The important printouts are the number of mutual web page linkages, the number of involvements per web page as outbound up to a maximum of the first 100 web pages, the n chosen top counted web pages and the time measurements of the functions.

The terminal input for this program is the text filename, it could be any web graph text file we want but we only use the already mentioned text files here, the specified format to be used and the number of top web pages we want to print for the CRS format. The CRS format is faster than the 2D table format, that is why we only use the **top_n_webpages** function with the CRS format.

3.2.3 Test_functions_OMP

This program does the exact same as the **Test_functions** program, except that now we use OpenMP to parallelize and time the functions. The reading functions are not parallelized, only the three other functions ending with _OMP. The printed results should yield the exact same solutions, only with faster time measured. We look at how the time measurements vary when we use different number of OpenMP threads as well.

4 Time measurement

The timing of the serial code is done with the *clock()* function from the *<time.h>* library, while the parallel code is timed with the *omp_get_wtime()* function from the *<omp.h>* library. We only study the time measurements, since the functions are made to give the same solutions for serial and parallel codes.

In Table 1 we see average times of the functions for both serial and parallel code when reading the **p2p-Gnutella05.txt** text file. The parallel codes are done with 8 threads for counting and sorting the web pages in this case. Here we see that the reading of the file are almost the same as expected, since the reading are not parallelized but still vary a little for each time we run our program. This is the same for both 2D table and CRS format for serial and parallel codes. For the 2D table counting of mutual linkages we get an approximate speed up of 3.55 of the average times from serial to parallel. For the CRS format, the used text file is so small that we cannot see any time difference here when counting the total mutual linkages. For the sorting of the top mutual linked web pages we get an approximate speed up of 7 from serial to parallel.

To really test the parallelization of the CRS format we use the much larger web graph text file **web-NotreDame.txt**. The time measurements of the parallelized function count_mutual_links2_OMP and top_n_webpages_OMP can be seen in Figure 1 and Figure 2, respectively. There we see that the counting seems to use the shortest amount of CPU time at 3 threads, but there the time measured of the sorting is not so low. The best overall parallelized time seems to be around 8 threads, which seems to fit since the

used computer has 8 virtual cores. In Table 2 we see the average time measurement of the CRS format functions for using both serial and parallelized implementation, with the use of 8 number of threads for the parallelized codes. The counting function is not much faster, but the sorting of the top involved web pages for mutual linkages gets a lot faster.

Functions	Serial [ms]	Parallel [ms]
read_graph_from_file1	32	28
read_graph_from_file2	13	11
count_mutual_links1	515	145
count_mutual_links2	<1	<1
top_n_webpages	7	1

Table 1: Table of the average times (in ms) for reading, counting and sorting the **p2p-Gnutella05.txt** text file with the functions with both serial and parallel implementation.

Functions	Serial [ms]	Parallel [ms]
read_graph_from_file2	386	381
count_mutual_links2	3	2
top_n_webpages	10	1

Table 2: Table of the average times (in ms) for reading, counting and sorting the **web-NotreDame.txt** text file with the CRS format in both serial and parallelized implementation.

References

[1] IN3200/IN4200 Home Exam 1, Spring 2020. https://www.uio.no/studier/emner/matnat/ifi/IN3200/v20/teaching-material/in3200_in4200_home_exam1_v20.pdf.

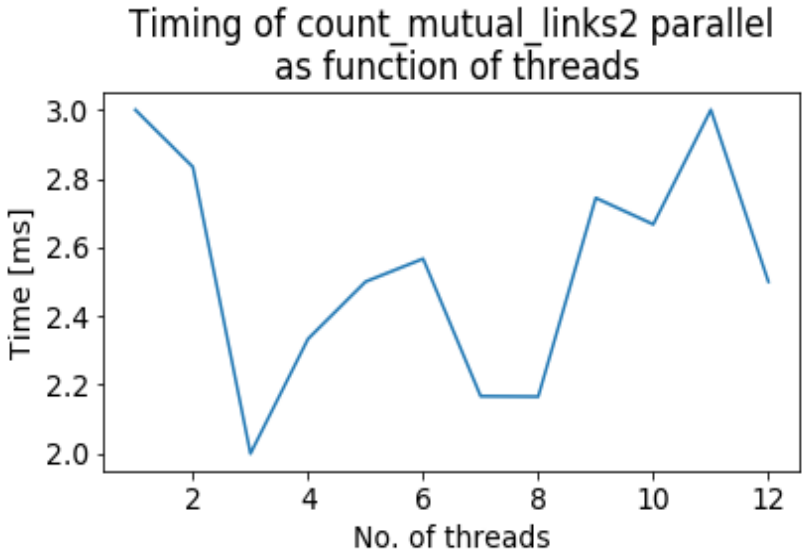

Figure 1: Time measurements of count_mutual_links2_OMP as function of the number of threads used.

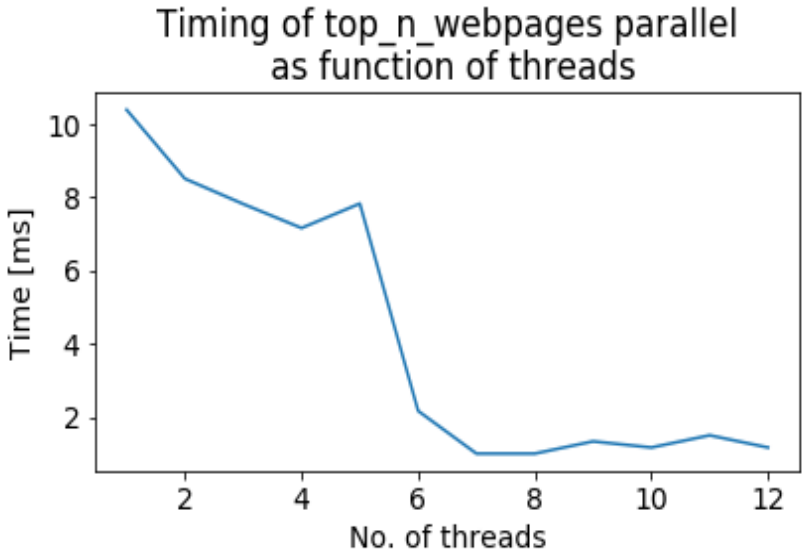

Figure 2: Time measurements of top_n_webpages_OMP as function of the number of threads used.