Home exam 1 - IN3200

Candidate nr: 15129
(Dated: March 12, 2020)

This is an abstract

I. INTRODUCTION

In this report we look into the main algorithmic aspects of the code implementations and present time measurements of the serial and parallelized codes.

II. METHODS

Webgraphs

Webgraphs describe hyperlinks between to webpages i and j. Each webpage is represented by a node and the hyperlink between them is known as an edge. In this report I'll define N as the number of nodes and N_{links} as the number of edges in a *directed* webgraph. By directed, we mean that the hyperlink $j \to i$ is distinct from $j \to i$.

Hyperlink matrix representation of webgraphs

Webgraphs can be represented by what is called a hyperlink matrix A. An element $A_{ij} = 1$ if there exists a hyperlink $i \to j$. If no such link exists, $A_{ij} = 0$.

Compressed row storage (CRS) of the hyperlink matrix

For large webgraphs, it's expected that the hyperlink matrix will consist mainly of zeros. Since the storage in the matrix format scales with N^2 where N is the number of nodes, storing the data becomes a bottleneck. The CRS format is based on to different arrays, a row pointer storing how many row elements that accumulates of a set of rows n and column index array that stores the column index for a given element. Let r denote the row pointer. The general form of the row pointer is

$$r = (0, r_0, r_0 + r_1, ..., r_0 + r_1 + \cdots + r_{N-1}), \quad (1)$$

where r_i is the number of row elements of row i. The row pointer thus has length $N + 1$. The column index array is of length N_{links} and stores the column indices for each row. To extract how many row elements there are row i, we take the difference $r_i - r_{i-1}$.

Mutual web linkages and number of involvements

Mutual web linkages is defined as the number of times to outbound nodes i and j are directly linked to a inbound node k, with $i \neq j \neq k$. That is, it's a count of how many times the exists a hyperlink $i \to k$ and $j \to k$ simultaneously. The *number of involvements* a given node has is defined as the number of times a given node is involved as an outbound node. That is how many times a node i is involved in $i \to k$ with $j \to k$.

Reading the Webgraph from file

1. *Hyperlink matrix storage*

To extract the data to store in a hyperlink matrix is straight forward. The following code snippet shows how I did it.

```
int FromNodeId, ToNodeId;
for (int k = 0; k < N_links; k++){
  fscanf(fp, "%d %d", &FromNodeId, &ToNodeId);
  (*table2D)[ToNodeId][FromNodeId] = (char) 1;
}
```

2. *CRS storage*

To extract the data from the file into CRS storage, we first must sort the arrays. I chose to sort this using the *shellsort* algorithm with gap $= N/2$. The following code snippet shows the implementation.

```
int tmp1, tmp2, i, j, gap;
for (gap = *N_links/2; gap > 0; gap /= 2){
  for (i = gap; i < *N_links; i++){
  tmp1 = row_elems[i];
  tmp2 = (*col_idx)[i];
  for (j = i; j >= gap && row_elems[j-gap] >
      tmp1; j -= gap){
    row_elems[j] = row_elems[j-gap];
    (*col_idx)[j] = (*col_idx)[j-gap];
  }
  row_elems[j] = tmp1;
  (*col_idx)[j] = tmp2;
  }
}
```

Here, row_elems is just there to temporarily store row node ids. The row pointer is simply made using an array counting how many elements each row has consistent with eq. (1). The following code demonstrates this.

```
*row_ptr = (int*)calloc(*N+1, sizeof(int*));
int count = 0;
for (int i = 0; i < *N; i++){
  count += row_count[i];
  (*row_ptr)[i+1] = count;
}
```

Counting mutual web links

Counting mutual web links with the hyperlink matrix

To count the number of web links

III. RESULTS

Timing of serial codes

The measured time of the serial implementations of the various functions are shown in table I.

Function name	Time in seconds
read_graph_from_file1	0.083591
count_mutual_links1	0.885690
read_graph_from_file2	0.356536
count_mutual_links2	0.001885
top_n_webpages	0.019348

TABLE I. The table shows the measured time using clock() from the Ctime-library. read_graph_from_file1 and count_mutual_links1 was applied to a web-graph containing $N = 10000$ nodes and $N_{\text{links}} = 37841$ edges as found in the file test_webpages.txt. The data in this file was extracted from web-NotreDame.txt. read_graph_from_file2 and count_mutual_links2 was applied directly to the web-graph contained in web-NotreDame.txt. This file contained $N = 325729$ nodes and $N_{\text{links}} = 1479143$ edges.

Parallelized version of count_mutual_links1

Using OpenMP to parallelize count_mutual_links1 gave the results shown in figure 1

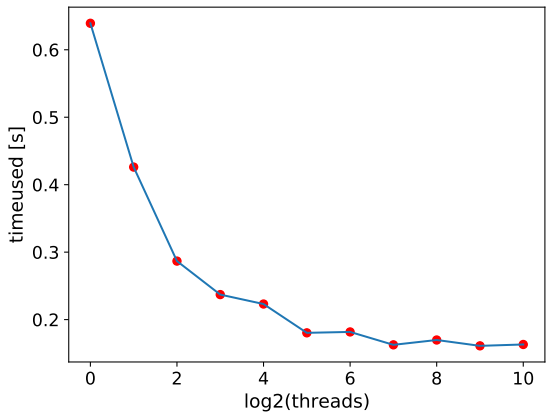

FIG. 1. The figure shows the time used in seconds by count_mutual_links1 on a webgraph consisting of $N = 50000$ nodes and $N_{\text{links}} = 146823$ edges. The webgraph was extracted from web-NotreDame.txt. The red points show the actual measured datapoints.

Parallelized version of count_mutual_links2

Using OpenMP to parallelize count_mutual_links2 and measuring the time used by the function for different number of threads yielded the results shown in figure 2.

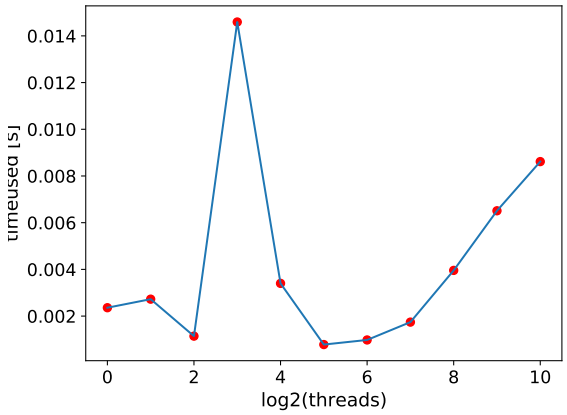

FIG. 2. The figure shows the time used in seconds by count_mutual_links2 on a webgraph consisting of $N = 325729$ nodes and $N_{\text{links}} = 1479143$ edges. The webgraph is found in the file web-NotreDame.txt

IV. DISCUSSION

V. CONCLUSION