Report – IN3200 Partial Exam, Spring 2019

Candidate No. 15220

April 4, 2019

Abstract

The main point of the program made in this project, is to list the top n webpages of a given web graph using the **PageRank algorithm**.

Introduction

A lot of information about the PageRank algorithm can be found online. The details necessary to understand this report are provided in the Partial Exam.

The program has been written in C. This makes it fairly straightforward to parallelize parts of the code using Open MP. In order for this to work, our system must have shared memory.

Given a web graph, a square hyperlink matrix can be set up. These are usually sparse (i.e. most elements are zero), giving huge potential for optimized storage. The storage scheme used in this project is the *Compressed Row Storage* (CRS). Detailed information about CRS is available online. A good place to start is Wikipedia.

The functions

The core of the program consists of three functions: `read_graph_from_file`, `PageRank_iterations` and `top_n_webpages`.

`read_graph_from_file` stores a web graph in the CRS format. It also stores information about dangling webpages. It takes a string (the filename of the file containing the webgraph) as its only input and returns a CRS struct.

The CRS struct is defined as folows:

```
struct CRS
{
        int *row_ptr;
        int *col_idx;
        double *vals;
        int nodes;
        int nz;

        int *dangling_idx;
        int dangling_count;
};
```

The three first entries should be self-explanatory for anyone familiar with CRS. Wise or not, the number of rows is stored as `nodes` (fourth entry), and the number of non-zero elements are stored as `nz` (fifth entry). The second to last entry holds the indices of the webpages with no outgoing links, so-called *dangling webpages*, and the last entry simply holds the number of dangling webpages.

`PageRank_iterations` implements the iterative procedure of the PageRank algorithm – please refer to the Partial Exam and comments in the code for details. It returns a converged PageRank score vector **x** of type double*. Its input arguments are the damping constant `d`, the convergence threshold ε (eps) and a pointer to the CRS Struct made in the previous function.

`top_n_webpages` prints the top `n` webpages of a converged PageRank score vector along with their scores in the terminal. In this context the topmost webpages are those with the highest scores. The function does not return anything. It takes the number of webpages to display `n`, a converged PageRank vector **x** and the length of that vector `nodes` as input arguments.

Time measurements and hardware and compiler information

The time measurements are presented in table 1. The numbers were generated by running `time.exe`. More on this in the next section. The damping constant used was 0.85, the convergence threshold was 1e-15, the web graph was web-NotreDame.txt. These measurements give a rough idea of

No. of threads	Time in seconds
1	1.423
2	0.870
3	0.691
4	0.562
5	0.935
6	0.903
7	0.742
8	0.692
12	0.865
24	1.210

Table 1: Time measurements of `PageRank_iterations`.

how a varying number of threads affects performance. The measurements were done quick and dirty, so the numbers are perhaps not exact to three decimal places as presented. The hardware information (Dell XPS 13, 8 GB RAM) is presented here:

```
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                8
On-line CPU(s) list:   0-7
Thread(s) per core:    2
Core(s) per socket:    4
Socket(s):             1
NUMA node(s):          1
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 142
Model name:            Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz
Stepping:              10
CPU MHz:               874.571
CPU max MHz:           3400,0000
CPU min MHz:           400,0000
BogoMIPS:              3600.00
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
```

```
L2 cache:                256K
L3 cache:                6144K
NUMA node0 CPU(s):       0-7
```

Compiler: gcc (Ubuntu 7.3.0-27ubuntu1 18.04) 7.3.0

Comments on the code

While writing the code, a program `test.exe` was used to verify the results. `test.exe` is compiled from `PE_test_15220.c`. This `.c`-file is included for the sake of completion, but keep in mind that it is not intended for others to read and understand outright.

To do the time measurements with a varying number of OpenMP threads, a program `time.exe` compiled from `PE_time_15220.c` was used. Here, the function `PageRank_iterations` is called a 15 times and an average time is given. This was done several times, varying the number of threads from one up to eight. Twelve and 24 threads are also included. Because the Partial Exam did not request a non-OpenMP version of the function, measurements of such a function are not included.

A few comments on the details of the implentation of the functions are perhaps in order. In general, a sane and efficient implementation has been pursued to the best of my ability.

In `read_graph_from_file` the web graph is traversed twice: once for counting the number of outbound and inbound links from and to each webpage, then a second time to store the correct values in the CRS scheme. This seems sane.

In `PageRank_iterations` a lot of work is done in parallel. Using OpenMP, it is fairly simple to handle most issues that arises when working with multiple threads, e.g. the issue of race conditions. I believe there are no revolutionary optimizations in the code submitted, rather a sane straightforward implementation.

The same goes for `top_n_webpages`; nothing out of the ordinary here.

Concluding remarks

There are probably as many solutions to this project as there are people on earth, a lot of which are probably faster than the one submitted here. That

said, the speed of this implementation is quite possibly sufficient for a lot of use cases.