Partial Exam: IN4200

Candidate number: 15301

April 3, 2019

1 Program structure

The program is build in three parts:

- Reading in a graph file and setting up the spare matrix of nodes and edges.
- Calculating the PageRank score for each site (also called node).
- Listing the top N sites.

Check the header file for explenation of functions doing each part.

2 Timing

The program was run on a system with AMD Ryzen 1700 @ 3.8 GHz with 8 cores/16 threads as the CPU, and DDR4 RAM with speed set to 2666 GT/s. The compiler was gcc with the optimazation flag -O3 used. Other flags used was -fopenmp for OpenMP, -lm for math functions, -std=c99 to specify the version of C used to C99.

OpenMP was used for the parallelization of the functions. For the PageRank iterations the sparse matrix multiplication used a dynamic scheduling with intermediate chunk size. This due to the different number occupied siters in the each row in the sparse matrix, which makes a static schedule load imbalanced. A chunk size of 1024 was choosen after testing for the balance between big scheduling overhead with small chunks sizes, and load imbalance with big chunk sizes.

In the sorting for the listing of the top N sites, the parallel tasking was only done if the number of indices in the section to be sorted in one of the recursive functions calls was bigger than a minimum found by numerical testing (about 10000). For cases where it wasn't the seirial quicksort algortihm was faster on that section.

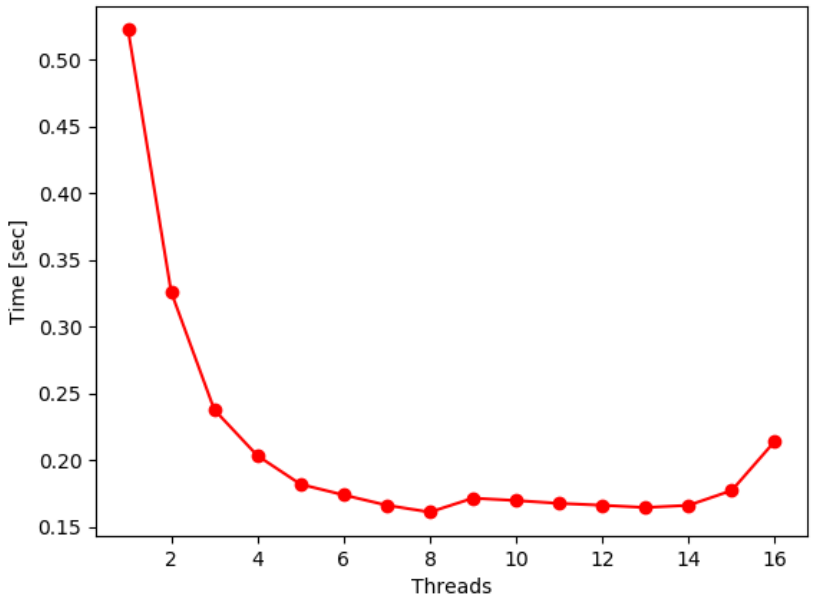

Figure 1: Timeing of the PageRank computation.

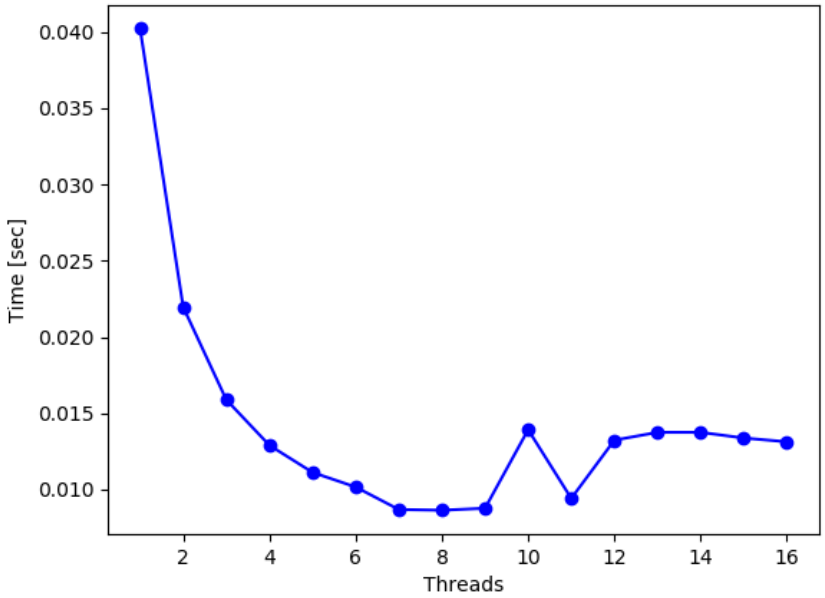

Figure 2: Timeing of the top N listing. Both functions maxes out in speed at about the number of physical cores. SMT doesn't seem to give a speed boost.

3 Dicussion

The fraction of peak performance of a code with balance B_c and a machine with balance B_m is:

$$l = \min\left(1, \frac{B_m}{B_c}\right) \tag{1}$$

where l is called the lightspeed of a loop which gives the performance:

$$P = P_{max} l = \min\left(P_{max}, \frac{b_{max}}{B_c}\right) \tag{2}$$

Here b_{max} is maximum bandwidth and P_{max}, peak performance of the machine.

A CRS-matrix multiply has $B_c = 5/4$ (4 reads and 1 write for 4 floating point operations done). For the computation of the scalar value of the dangling indices W^{k-1} (for the k-th iteration), we have a $B_c \approx 2$. Here it is assumed that the writing of W^{k-1} is not done untill the loop is done so that one has two reads for each addition.

```
#pragma omp for reduction(+: dangling_score)
for (k = 0; k < crs.len_dangling; k++) {
    dangling_score += pagerank_score_past[crs.dangling[k]];
}
```

This gives for one iteration $5/4 \leq B_c < 2$. Plugging this in equation (2) gives us the maximum achievable computation. Converting RAM speed to MB/s and CPU clock rate to $GFLOPS/s$ we get $B_m = \frac{21333.33MB/s}{16*16*3.8GFLOPS/sec} \approx 0.022$. Which mean the lightspeed of the iteartion loop is a best $l \approx 0.018$. This mean the best performance possible (no dangling site) for this algorithm is $P = P_{max} l \approx 17.5 GFLOPS/s$. The `web-NotreDame.txt` graph file has 1.5 million edges. A the best performance possible this would for 173 iterations ($173 * 5 * 1.5$ MFLOP) take ≈ 0.05 seconds. This is 3 times better than achived (and impossible since there is dangling sites).

In reality the computation of take alot more times since alot more read and writes are done to update the past vector and the updated vector with node scores. The number of dangling sites and the non-continous read of the score vector for computing the dangling score also slow down the computation.