IN 3200 – High Performance Computing

Partial Exam – Candidate 15219

Structure

The algorithm is divided into 3 parts:

1) Reading the text file.
2) Giving each web page a score.
3) Finding the top N web pages.

Reading the text file

I've implemented this function in the following three steps.

I) Go through the text file, count the occurrences of FromNodeId and ToNodeId.

II) Go through the file once more. Use the information found in the previous pass to create the CRS in addition to a "boolean" array of which nodes are dangling web pages. The CRS exists as three 1-Dimensional arrays; row_ptr, col_idx and val.

III) Return the CRS and the Dangling array back to main.

Giving each web page a score

I've included a bunch of sequential functions for a deeper understanding of the different algorithms. These were the initial implementation before i parallelized using OpenMP.

$$\mathbf{x}^k = \frac{\left(1 - d + d \cdot W^{k-1}\right)}{N} \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} + d \cdot \mathbf{A}\mathbf{x}^{k-1}$$

In this function – go through a number of iterations of the calculation of X^k
until we've reached a threshold. For each iteration:

I) Begin by finding the "W", which is equal to the left side of the equation for X^k. Add the result to x_new[]

II) Compute the sparse matrix multiplication, Ax^{k-1}. When done, multiply each row by the damping factor. Sum the result together with part I) to complete the calculation of X^k.

III) Pointer-swapping from x[] and x_new[]. Also check if we have converged yet. Current Iteration done. If we haven't converged yet, repeat from step I).

Time Measurements of PageRank Algorithm

Benchmarking was done on my laptop consisting of a:
i7-7500U CPU @ 2.70GHz - Dual-core / 2 threads per core.
For a total of 4 Threads. My memory includes 8GB of RAM, running at 1867Mhz. In addition to L1 cache of 128kB, L2 cache 512kB and L3 cache 4MB.
Compiled with the gcc- compiler.

```
Class           Description
=========================
memory          8GiB System Memory
memory          4GiB Row of chips LPDDR3 Synchronous 1867 MHz
memory          [empty]
memory          4GiB Row of chips LPDDR3 Synchronous 1867 MHz
memory          [empty]
memory          128KiB L1 cache
memory          512KiB L2 cache
memory          4MiB L3 cache
```

All runs done with d=0.85, eps= 0.00000000000000000001

Threads	8-node example	100-node example	Web-NotreDame.txt
1	0.145ms	1.451ms	1.635s
2	0.310ms	1.416ms	0.927s
3	0.340ms	1.390ms	1.039s
4	0.600ms	1.562ms	0.868s
Iterations needed:	9	73	178

As we can see from the results above, the overhead of both creation of threads as well as communication and synchronization between threads shows how running the code sequential for small web-graphs proves more efficient. For larger web-graphs such as the

NotreDame text file, the parallel solution proves superior. I have managed to get a speedup of approximately 1.88 from 1 thread to 4 threads on the NotreDame file. Here we can see how Amdahls law comes into play, even though we have 4 times as many threads, the speedup is not 4-times as fast. This can be explained due to the overhead mentioned above, in addition to the fact that there is still some serial work that has to be done.

Finding the top N web pages.

I realized it is not necessary to sort all web pages if we want to find the top N pages.
A more efficient way of doing this can be done by the so called top-K-sort. These steps summarize how I have implemented this function.

I) Sort the first K web pages. When sorting we also have to keep track of the web pages' initial number, or ID if you'd like.

II) For each next web page (k+1 … k+(n-k)) , check if this web page has a higher score than the lowest top-k- pages. If yes, swap positions and re-sort the top-k- results.

III) Print the top N pages found. Also return an array of the top scores.

Compilation

I have created a makefile to compile the program. Compile by typing ‘make’. This will create a file called run. The program can now be run by the command ./run

How to run

Run the program by ./run 'file_name' 'damping_factor' 'epsilon' 'top_n_webpages' ‘threads’
If you want to run the NotreDame file with a damping factor of 0.85, epsilon of 0.00001 ,find the top 10 web pages and use all avaible threads on your current computer, you would run:

./run web-NotreDame.txt 0.85 0.00001 10 0

END.