CO316 - Computer Architecture Lab

End Semester Examination Report on

'Betweenness Centrality of Large Sparse Graphs'

Submitted By:

Mahir Jain (16CO123) and Suraj Singh (16CO146)

Observations:

GPU used: Nvidia Tesla K80 (Through Google Colab)

CPU used: Intel core-i5 8250

Input (Number of Nodes)	BC Value	Parallel Execution Time in seconds	Serial Execution Time in Seconds*	Speed Up (CPUTime ÷ GPUTIme)
10	15	0.00175	0.030694	17.5394285714
10 ^ 2	451.723053	0.001125	0.118475	105.311111111
10 ^ 3	13682.895508	0.008973	2.316074	258.115903265
10 ^ 4	255227.093750	26.391409	296.771326	11.2449974156
10 ^ 5	N/A **	2635.802207	N/A **	N/A **

*We Used the *time.h* library to compute the execution times of our program on inputs of different size.

** We made a small mistake while setting our 10^5 script to run. We forgot to find the max BC value of the given graph and hence could not get the value. We only have about 15 minutes to the submission deadline and hence cannot execute the program once again.

Note - 2635 seconds is about ~45 minutes. Running the serial version with a 0.1x speedup for CPU would take about 5.5 hours and hence we could not get the corresponding CPU execution time for 10^5.

As can be seen, for 10^4, our implemented program gives ~10x speedup which matches up with what the paper claims.

Implementation Details:

We used **our own graphs** as there was an issue with the common graphs being produced. (Files very large, could not be uploaded to Colab). The dataset will be sent along with this report for the results to be reproduced. The graphs were of the CSR format, as text files, and not the binary files that are present on the college server.

The main reason for the above was the need to run code on Google Colab, and we are limited by the amount of Drive space that is available to us.

The graphs have number of edges as the order of the number of nodes. In most cases, to be precise, number of edges was roughly 3 times the number of nodes.

Instructions to Run (Parallel):

1. Open the .cu file
2. In the preprocessor directives with #define, set the values of r_size and c_size to the number of lines in the Column Offset and Adjacency List files respectively. (These would be 10 and 28 as per the example given on the Nvidia CUDA blog over here)
3. In main(), ensure that the file handling statements are accessing the correct filenames.
4. Use nvcc to compile the source.
5. Execute the binary.

Approaches Taken:

1. After reading about BC, we first wanted to make sure we understood the concept fully and hence, decided to implement the serial version of the program first. We followed this resource to understand Brandes

algorithm fully. It was very helpful because it started with a naive Floyd-Warshall method and incrementally added steps to arrive at Brandes algorithm. Now, having understood the algorithm, we implemented it on the graph given in the nvidia blog article and it worked. Since we implemented it, we decided to run the bigger graphs on it as well to compute speed ups.

2. We were following the approach taken in the paper sent to us in the mail titled 'Accelerating GPU Betweenness Centrality' to implement the **work efficient** approach to the problem . We realised using the second queue was slightly redundant, and hence we opted to use only one queue(Q) as our "stack" (from the serial approach) and used the ends[] array as our pointers to different frontiers in our Q. We even came across a situation where we had to set barriers in the same fashion that we did in the *Ocean Kernel* program taught to us in the Systems Programming course last semester. However, we were encountering a few bugs, and we weren't sure how all threads could add to the stack with the appropriate offsets and considering the time crunch, we abandoned this approach and moved to the same one used in the paper. We feel with a little more time, we could get this to work and hence have attached the code for this in a file called "onequeue.cu".
3. The approach that finally worked for us was following the aforementioned paper to the T. The only problem was that we didn't understand what things like "S" that were used in the paper stood for, how they should be initialized etc. After some extensive searching, we found this resource. The paper has explained how exactly each variable and array needs to be initialized and is definitely more intuitive to read and we got this approach to work.

Problems we faced with the algorithm itself:

1. Brandes algorithm is designed for directed graphs. Betweenness Centrality should only be counted from paths from node s to t through

v and NOT t to s through v as well. Hence, when we add delta[w] files to cb[w], we needed to change it to cb[w] += delta[w]/2.

2. The algorithm breaks out of the while loop by setting the depth to d[S[S len − 1]] - 1, however it should actually be set to d[S[S len − 1]] only.

Acknowledgement:

We are grateful to Prof. Basavaraj Talawar for giving us such a unique opportunity. It felt like a hackathon and the problem statement was very interesting. We are looking forward to more such challenges in the future.