CS 525   Project 2 Report

Name: Ali Necat Karakuloğlu
ID: 22101543

Implementation Details

i)*Serial*
For this part, I implemented the algorithm given in the project documentation. Then, I tested the algorithm with small graph that is provided in the project folder. I compared the results and verified the correctness. Furthermore, I have added a loop breaking mechanism to the algorithm to make early termination and an overflow mechanism (INT_MAX + number = INT_MAX).

ii)*Parallel*
Before implementing the parallel version of Single Source Shortest Path (SSSP) algorithm, I partitioned the data between processes. At the first step, I parititoned the vertices of graph between processors. Note that, this partitioning is even. Namely, each processor has the same number of piece from rows(vertices) array. To use the scatter function of MPI, I broadcasted the required data to other processors from master node. The data is the number of nodes and edges of the graph. Using this data, each process generates a buffer to store vertices. After scattering vertices, Each process generates arrays for edge partition and vertice partition using the boundaries in the rows partition (vertice partition). This is not a static partition. In other words, the sizes of the arrays determined by the difference between final and first entry of vertice partition. Then, using this buffers, edges array and weights array is scattered by the master process.

Since each process has its data, we can move on with the parallelization of the SSSP algorithm. As mentioned in the project documentation, getting local results and combining them at the same array is the key point because, everything else is almost the same with serial algorithm. For the combination of partial results, I used allgather operation. After getting the combined result, compared it with D array. If they are the same, the execution is terminated.

Analysis

I expected to see speedup parallel with the processor number. However, the parallel version didn’t meet my expectations. As you will see in the next section, the maximum speedup was 2. In most of the cases, communication cost exceeds computation cost. Therefore, the speedup is not high as expected.

Experiments

To evaluate the algorithm, I used three datasets [1][2][3]. Execution time and speedup vs. processors data are provided in Figure 1-3. Furthermore, the overhead for each graph is given in Figure 4. I couldn’t convert the big graphs with 1.5M edges (or larger) to CSC format. The largest graph is Notre-Dame web network graph. It has 1.49M edges and 325K nodes. Highest speedup is achieved with Amazon product co-purchasing network. The speedup is just above 2 as can be seen from Figure 1.

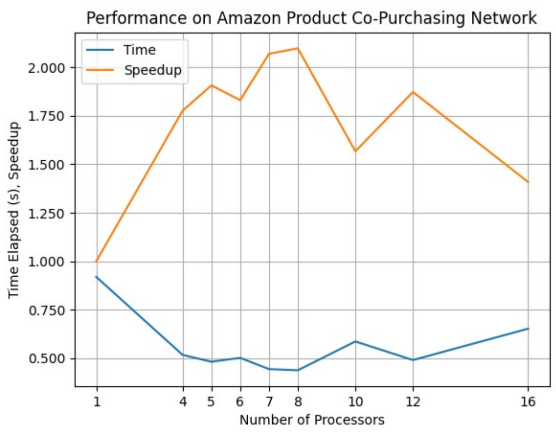

Figure 1: Performance Graph for Amazon co-Purchasing Network

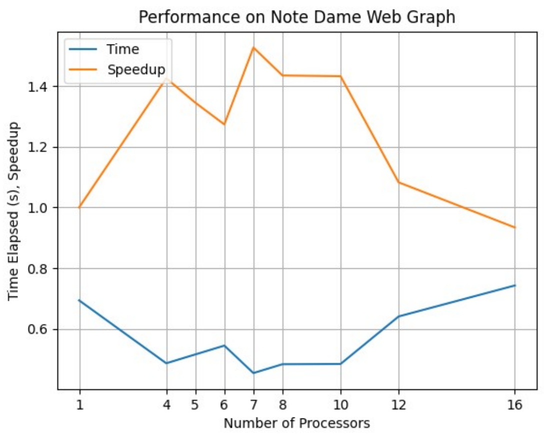

Figure 2:Performance Graph for Notre Dame Web Graph

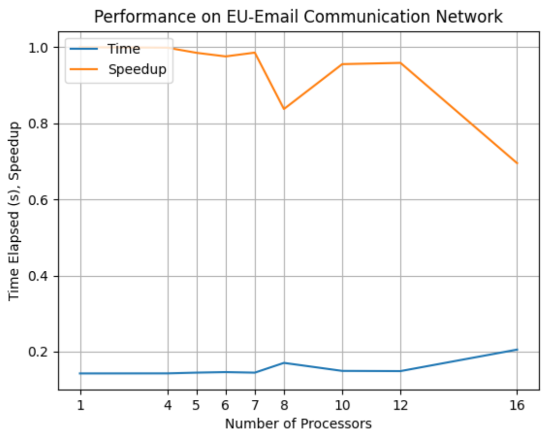

Figure 3:Performance Graph for EU-Email Communication Network

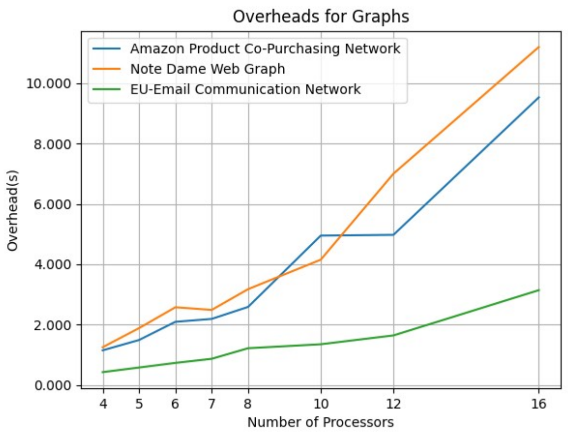

Figure 4: Overhead vs. Number of Processors Graph

Since the data distribution is mostly irregular and the overhead increases significantly with the increasing number of processors, the algorithm is not efficient and it can’t be scaled. To make the calculation more efficient and scalable, we need to change the algorithm.

Discussion

Parallel implementation didn’t improve the EU-email communication network graph at all. On the other hand, there is an improvement on Amazon graph and Notre Dame graph. However, this improvement is limited. The communication overhead and lack of connections between the nodes (connectivity of the nodes are limited) has affected the overall result as can be seen from the previous part. Furthermore, the load imbalance between processors effects the overall performance significantly. One processor may have much more less number of edges assigned to it. This one processor therefore needs to wait for others to finish. Since we are given the problem as it is (we can’t change the features and topology of the graph), we need to answer the load imbalance. One way that I think it might be helpful is to change the graph representation. If we consider this calculation as a dense matrix-vector multiplication, then there is no load imbalance. We can evenly divide the workload between processes. However, a dense matrix multiplication algorithm has lots of redundant operation. To solve this issue, we may represent the graph in bitmap format. The bitmap format has two data structures. One of them is binary matrix that consists of 1s and 0s. Another one is the array that nonzero data is held. By partitioning a binary matrix, we can map the operation to dense multiplication with less effort. I give a research paper as reference [4].

Note: In some of the graphs, the number of edges or nodes wasn’t evenly divisible with number of processors. Therefore, I added padding. Namely, I added connections or new nodes to the graph to make numbers fit. Otherwise, the program gives segmentation fault.

Platform Specifications:

◦ Operating System: Ubuntu 20.04.3 LTS
◦ CPU: AMD Ryzen 7 3700X 8-Core Processor
◦ Number of Logical Cores: 16
◦ RAM: Corsair Vengeance 8 GB 3200 MHz DDR4, x2
◦ HDD/SSD: SanDisk SSD PLUS 120GB
◦ Compiler: g++ (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0 / mpicc

References:

[1] J. Leskovec, L. Adamic and B. Adamic. The Dynamics of Viral Marketing. ACM Transactions on the Web (ACM TWEB), 1(1), 2007.
(Website:https://snap.stanford.edu/data/amazon0302.html)

[2]R. Albert, H. Jeong, A.-L. Barabasi. Diameter of the World-Wide Web. Nature, 1999.
(Website:https://snap.stanford.edu/data/web-NotreDame.html)

[3]J. Leskovec, J. Kleinberg and C. Faloutsos. Graph Evolution: Densification and Shrinking Diameters. ACM Transactions on Knowledge Discovery from Data (ACM TKDD), 1(1), 2007.
(Website:https://snap.stanford.edu/data/email-EuAll.html)

[4]Jianting Zhang and Le Gruenwald. 2018. Regularizing irregularity: bitmap-based and portable sparse matrix multiplication for graph data on GPUs. In Proceedings of the 1st ACM SIGMOD Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA) (GRADES-NDA '18). Association for Computing Machinery, New York, NY, USA, Article 4, 1–8. DOI:https://doi.org/10.1145/3210259.3210263