CS121 Parallel Computing Lab 1
Breadth First Search in OpenMP

Problem description

In this lab you will implement a parallel breadth first search algorithm in OpenMP. BFS has a large number of applications, and accelerating BFS can improve the performance of a range of downstream tasks. In addition, while parallelizing BFS is straightforward, achieving high performance requires a well thought out design. The input to your program will be an undirected graph and a source node in the graph, and the output requires finding all the nodes which are reachable from the source node, as well as their distances and paths to the source.

Specifically, your program should accept as input a graph specified in the Matrix Market format, and the ID of a source node in the graph. Matrix Market is similar to the COO format described in class, and information about it is available at https://math.nist.gov/MatrixMarket/formats.html. The output is a file consisting of two parts. The first part is a single line containing one integer n, equal to the number of nodes in the connected component containing the source node. The next part consists of n lines, each containing 3 values <*node id*> <space> <*distance*> <space> <*parent*>. The first value is the ID of a node v (each line contains a different node). The second value *distance* is the minimum number of hops from the source node to v. The third value *parent* is the parent of v in a BFS tree rooted at the source. Note that the parent may not be unique, as there may be multiple BFS trees from the source.

Experiments

You will test the performance of your algorithm by benchmarking it on the following graphs. The first four graphs can be downloaded from the SNAP dataset at https://snap.stanford.edu/data/index.html. The RMAT graphs can be generated using the snap.py package and its GenRMat function, with the values of parameters a, b, c as shown below; see https://snap.stanford.edu/snappy/doc/reference/GenRMat.html. If GenRMAT works too slowly, you can use a parallel RMAT generator available at https://github.com/farkhor/PaRMAT. Note that you may need to convert the graphs into the Matrix Market format before using them. Conversion functions are available in Matlab or Python packages such as networkx.

Graph	Nodes	Edges	Notes
web-Stanford	281,903	2,312,497	Web graph of Stanford.edu
roadNet-CA	1,965,206	2,766,607	California road network
com-Orkut	3,072,441	117,185,083	Orkut online social network
soc-LiveJournal1	4,847,571	68,993,773	Livejournal online social network
RMAT 1	100,000,000	1,000,000,000	$a = 0.3, b = 0.25, c = 0.25$
RMAT 2	100,000,000	1,000,000,000	$a = 0.45, b = 0.25, c = 0.15$

RMAT 3	100,000,000	1,000,000,000	$a = 0.57, b = 0.19, c = 0.19$

1. First benchmark the single-threaded performance of your algorithm on each graph by performing BFS from a random source node. Report your performance in terms of MTEPS (millions of edges traversed per second). In particular, count the number of edges m in the connected component containing the source node which your algorithm returns. Then, if the running time for the algorithm was t seconds, your performance is $m/(10^6 t)$ MTEPS.

   To obtain consistent results, choose 20 different source nodes for each graph and report the average performance and standard deviation. Exclude the time for loading the input and producing the output from your benchmarks.

2. Test the scalability of your algorithm by performing the same experiment as above, but using different numbers of threads, up to the limit on your machine.

Analyze the performance of your algorithm. If you initially fail to obtain good performance or scalability, optimize the algorithm to improve its efficiency. Your score on the lab depends on the design of your algorithm and its performance.

Submission

Prepare a report clearly describing your algorithm, your benchmarks, and the configuration of the machine you benchmarked on. Also describe any optimizations you performed or problems you encountered. Create scripts to easily compile your code and for running the experiments described above. Also provide a function to run your code using a user-specified graph stored as a file in Matrix Market format, so that we can test your program. Submit your report and code to the TA. The due date for the lab is **11:59pm, December 22, 2023**.