Advanced Programming Techniques

Parallel search

Recommended programming language: C++

Disclaimer: efficient implementations of graph data structures and algorithms can be already found in various dedicated libraries. The goal is to write your own code!

1) We shall use OpenMP in order to parallelize our codes.
   If you do not have GNU compiler already installed, then start doing so:

   https://gcc.gnu.org/mirrors.html

   Most recent versions of gcc already have OpenMP configured. You need to compile your codes wth the option `-fopenmp`.
   Run the following starting example from the tutorial in order to familiarize yourself with OpenMP annotations:

   ```
   <include omp.h>
    int main() {
      #pragma omp parallel
      {
         cout << ``Hello'';
         cout << `` word!'' << endl;
      }
   }
   ```

   - Modify the above code in order to also output the ID of the currently running thread (the latter can be accessed with the function `omp_get_thread_num()`).
   - You may also change the number of running threads by calling the dedicated function `omp_set_num_threads(int)`.

2) Parallelize your previous codes for BFS/DFS. Specifically, each time a vertex is visited, we remove it (in parallel) from the adjacency list of all its neighbours. For that, use the annotation `#pragma omp for` before the for loop. Using SNAP datasets, compare the performances obtained with your sequential code, as a function of the number of threads.

3) The BFS code is further modified by having two queues: the frontier, which contains the vertices of the current distance layer, and the next frontier, where we insert the vertices of the next layer. Doing so, we can also iterate in parallel on all vertices of the frontier. Note that it can lead to a vertex being inserted several times in the next frontier. Compare the performances of this new version with sequential BFS and the parallel version of question 2).

4) Implement the `Bag` data structure, discussed in class.

5) Using Bags, implement the fast parallel BFS discussed in class. Note that each time your halve the bag, your both threads need to execute in parallel a different code. Their respective next frontiers can be merged at the end of their execution by using the `reduction(op:obj)` annotation.