Parallel Programming 2020, Assignment 2: Parallel Minimum Spanning Tree

Deadline: Monday, November 16 before 23:59 hours.

In this second programming assignment, you will implement a parallel Minimum Spanning Tree (MST) algorithm. Your algorithm should be based on Borůvka's algorithm that was discussed in class. After implementation you will perform a small benchmark using three test graphs.

The assignment has to be completed individually. You are expected to hand in a tarball containing your source code and a report briefly describing your implementation of the algorithm and the benchmark results in PDF format. The assignments can be handed by e-mail to *hpc1-2020 (at) dvdzwaan (dot) com*.

1 Implementation

Your algorithm needs to be implemented in the C/C++ language, using MPI for interprocess communication. The target platform is the DAS-5 cluster. Parallelization is *only* to be done by distributing the work over different nodes. So, per node only one CPU core and one CPU thread will be used. The maximum degree of parallel is limited to 24. It is recommended to use the skeleton from assignment 1 as the starting point.

Note that your implementation should also handle disconnected graphs, resulting in multiple spanning trees. Also note that as output you should report the total weight of each minimum spinning tree as well as the edges which belong to the spanning tree. Edges with negative weights are expected to be handled correctly as well. A spanning tree may thus have a negative total weight.

2 Benchmarking

The benchmark needs to be performed on the DAS-5 system at Leiden University. You should benchmark with 2, 4, 8, 16 and 24 nodes to see how well the performance of your algorithm scales as more compute nodes are added. As test data, we will again use sparse matrices from the SuiteSparse Matrix Collection (`https://sparse.tamu.edu/`). The following matrices, that represent undirected weighted graphs, were selected:

Belcastro/mouse_gene (29M nnz)
GHS_psdef/ldoor (42M nnz)
Schenk/nlpkkt240 (760M nnz)

During code development you can use the smaller *Gaertner/nopoly* matrix for testing.

Include the obtained benchmark results in your report, together with a brief discussion.

3 Input & output

Your submission is expected to include a *Makefile* which will build and output an executable called "`mst`" in the *same directory* as the Makefile.

In order to easily benchmark your application, the executable is expected to take exactly one argument: the file name of a Matrix Market formatted file.

Your program is expected to output various numbers on stdout. First, the time taken to compute all spanning trees in seconds:

```
fprintf(stdout, "%.20f\n", elapsed_time.count());
```

Then, information about each spanning tree and the edges that are part of it should be output in the following format:

```
for (spanning_tree : spanning_trees) {
  fprintf(stdout, "%d", number of edges in spanning_tree);

  double total_tree_weight = 0.0;
  for (edge : edges in spanning_tree) {
    fprintf(stdout, "%d %d %.20f\n", edge node 1, edge node 2, edge weight);

    total_tree_weight += edge weight;
  }

  fprintf(stdout, "%.20f\n\n", total_tree_weight);
}
```

Nothing else should be written to stdout. Any output on *stderr* is ignored.

It is recommended to use *prun* to easily test your application: `prun -np 2 -script $PRUN_ETC/prun-openmpi `pwd`/mst some_matrix_market_file`. This will also be used to verify your submitted program works correctly. If you use a custom job script, mention this clearly in an additional *README* file.

4 Links

- DAS-5 website with information on job submission and starting MPI programs: `https://www.cs.vu.nl/das5/jobs.shtml`.
- Online MPI API documentation: `https://www.open-mpi.org/doc/v1.8/`. The API documentation is also available on DAS-5 in the form of man pages after executing the `module load openmpi/gcc` command.
- MPI tutorial from Lawrence Livermore National Laboratory (LANL): `https://computing.llnl.gov/tutorials/mpi/`.