# Project Q2: Graph Partitioning

## Project's summary

Many applications of computer science involve processing large graphs, and often these graphs need to be partitioned into pieces that do not overlap. Dividing a graph into p partitions is called p-way partitioning. This project requires implementing and comparing a sequential and a parallel version of a graph partitioning algorithm.

## Problem Definition

We consider graphs $G = (V, E)$ with two weight functions $W_1$: $V \rightarrow R$ mapping vertices to real-valued weights and $W_2$: $E \rightarrow R$ mapping edges to real-valued weights.
A p-way partitioning of G is a division of G into p sub-graphs in which the vertices in each subset do not overlap and satisfy specific properties:

- The sum of the weights of the nodes in each subgraph is balanced.
- The sum of the weights of the edges crossing between subsets is minimized.

Partitioning a graph has many applications in WEB search (Page Rank), placement and routing of VLSI circuits, social networks, etc.
Following the indicated references, this project implies.

- Reading large benchmark graphs from long-term memory.
- Implementing at least one sequential version of a partitioning algorithm.
- Implementing at least one parallel version of a partitioning algorithm.
- Comparing the sequential and the parallel version of the algorithm.

The comparison must consider the computation time and the memory usage of the main phases (at least, graph loading and path computation) and compare the sequential version with the parallel one with an increasing number of threads (i.e., 1, 2, 3, 4, etc.) on graphs of increasing size and complexity.
Optional: Compare the result of the written tool with publicly available applications such as hMetis, or the one available in Python in scikit-learn.

## Required Background

Background aspects are all covered within the following courses:

- Advanced problem-solving and programming classes (e.g., "Algorithm and Programming")
- "System and Device Programming" course, i.e., concurrent programming.

## Working Environment

Students can work on their laptop or desktop. Each group can decide whether to develop the application under a Unix-like or a Windows system. The implementation can be done in C, C++, or C++ using the boost library (for GPU optimizations). Parallel algorithms are described on the papers reported in the reference section. These works are available in the reference directory.

## Constraints

All projects must be delivered including the following material:

- The source C/C++ files.
- A README text file (written in plain ASCII) including a short "user manual", i.e., a document describing how to compile and run the program, under which system, which API, etc.
- A DOCUMENTATION text file (written in Word, Latex, Mark-down, etc.) including a short "designer manual", i.e., a document including:
    - All main design choices (how the reading part has been performed, how the data structure has been organized, how the parallelism has been designed, etc.)
    - The experimental evaluation of the tools, i.e., tables or graphics reporting a reasonable set of experimental results. The experimental evidence should include a comparison (in terms of memory and of elapsed time) between the original sequential version of the tool and the 2-3 parallel versions with different parallelization levels (i.e., with 1, 2, 4, 8, etc., threads) and different size of the input graph (up to millions of nodes).
- An OVERHEAD set (organized in PowerPoint or similar) to be used during the project discussion in the project evaluation phase.

## Contact

Prof. Stefano Quer (stefano.quer@polito.it).

## References

- See https://en.wikipedia.org/wiki/Graph_partition for a theoretical introduction on the topic.

- See references reported in the directory dedicated to this project for technical details.