
# How to compile

- In order to compile the entire project, you will use the make files, you need to do the command `make` in the main directory `tarjan/`. The binary files are created into the directory `tarjan/bin/`and subfolders containing in turn a bin folder.
- The makefile automatically searches for mpicc and nvcc in the system and only compiles the portions of the project that can be compiled in the current environment (e.g. if cuda is not installed in the system, the makefile won't build targets 'cuda' and 'cuda_texture' that need nvcc)
- You can also individually fill in a sub-section of the project using the make file in the specific sub-section.
- The `make clean` command deletes all files required for compilation (binary and object files).

# How to run

## Algorithms

### Sequential Tarjan Algorithms

- This algorithm performs sequential Tarjan on a given input graph from a binary file (.bin) following the graph.h library representation.
- To use such an algorithm, we run the program with one of the four possible optimisations (0,1,2 and 3) in the `./tarjan/bin/` folder after compilation, `./tarjan/bin/sequential_Ox.out` , x represents the chosen optimisation.
- The first input parameter is the graph on which we want to run this algorithm `./tarjan/data/seed.bin`.
- The second input parameter is the file that will contain the SCCs found `./tarjan/data/SSC_discovered.bin`.
- An example of using it in the root directory `./tarjan/` is as follows: `./bin/sequential_O0.out ./data/seed.bin ./data/SSC_discovered.bin`.

### Preprocess sequential Tarjan Algorithms

- This algorithm performs Tarjan's algorithm but with an antecedent step. Before passing the graph to Tarjan sequential we perform a preprocessing in which trivial SCCs (nodes that have an edge towards themselves) and nodes that cannot be part of SSCs, which are therefore useless to control (nodes without outgoing edges and nodes without incoming edges) are removed.
- To use this algorithm, we run the program with one of the four possible optimisations (0,1,2 and 3) in the `tarjan/bin/` folder after compilation, `./tarjan/bin/sequential_pre_Ox.out` , x represents the

chosen optimisation.
- The first input parameter is the graph on which we want to run this algorithm `./tarjan/data/seed.bin`.
- The second input parameter is the file that will contain the SCCs found `./tarjan/data/SSC_discovered.bin`.
- An example of using it in the root directory `./tarjan/` is as follows: `./bin/sequential_pre_O0.out ./data/seed.bin ./data/SSC_discovered.bin`.

## Parallel MPI Tarjan Algorithms

- This algorithm implements a version of a parallelization of Tarjan's algorithm using MPI.
- To use this algorithm, we launch the program with `mpiexec`.
- The first input parameter is the number of processes to instantiate `\-n y`, where y is the number of processes.
- The second input parameter is the binary file of the mpi program, with one of the four possible optimisations (0,1,2 and 3) in the `tarjan/bin/` folder after compilation, `./tarjan/bin/mpi_Ox.out` , x representing the chosen optimisation.
- The third input parameter is the graph on which we want to run this algorithm `./tarjan/data/seed.bin`.
- The fourth input parameter is the file that will contain the SCCs found `./tarjan/data/SSC_discovered.bin`.
- An example of using it in the root directory `./tarjan/` is as follows: `mpiexec -n 8 ./bin/mpi_O0.out ./data/seed.bin ./data/SSC_discovered.bin`.
- In order to execute the application on a cluster of nodes, a host file must be provided with the argument `--hostfile path_to_hostfile` (see MPICH docs).

## Parallel CUDA Global Memory Tarjan Algorithms

- This algorithm performs Tarjan's algorithm but with a preprocessing step performed by the GPU, using CUDA (nvcc v12.0). Before passing the graph to Tarjan sequential, a preprocessing is performed in which trivial SCCs (nodes that have an arc to themselves) and nodes that cannot be part of SSCs, which are therefore useless to check (nodes without outgoing arcs and nodes without incoming arcs) are removed. The data structures used for this GPU preprocessing phase are located on the Global Memory.
- To use this algorithm, we run the program with one of the four possible optimisations (0,1,2 and 3) in the `tarjan/bin/` folder following compilation, `./tarjan/bin/cuda_Ox.out` , x representing the chosen optimisation.
- The first input parameter is the graph on which we want to run this algorithm `./tarjan/data/seed.bin`.
- The second input parameter is the file that will contain the SCCs found `./tarjan/data/SSC_discovered.bin`.
- The third parameter indicates the total number of cuda threads created considering the fixed one-dimensional block size of 512 threads.
- An example of using it in the root directory `./tarjan/` is as follows: `./bin/cuda_O0.out ./data/seed.bin ./data/SSC_discovered.bin`.

## Parallel CUDA Texture Memory Tarjan Algorithms

- This algorithm performs Tarjan's algorithm but with a preprocessing step performed by the GPU, using CUDA (nvcc v12.0). Before passing the graph to Tarjan sequential, a preprocessing is performed in which trivial SCCs (nodes that have an arc to themselves) and nodes that cannot be part of SSCs, which are therefore useless to check (nodes without outgoing arcs and nodes without incoming arcs) are removed. The data structures used for this GPU preprocessing phase are located on the Texture Memory.
- To use this algorithm, we run the program with one of the four possible optimisations (0,1,2 and 3) in the `tarjan/bin/` folder after compilation, `./tarjan/bin/cuda_texture_Ox.out` , x representing the chosen optimisation.
- The first input parameter is the graph on which we want to run this algorithm `./tarjan/data/seed.bin`.
- The second input parameter is the file that will contain the SCCs found `./tarjan/data/SSC_discovered.bin`.
- The third parameter indicates the total number of cuda threads created considering the fixed one-dimensional block size of 512 threads.
- An example of using it in the root directory `./tarjan/` is as follows: `./bin/cuda_texture_O0.out ./data/seed.bin ./data/SSC_discovered.bin`.

### Parallel CUDA Global Memory and MPI Tarjan Algorithms

- idea algoritmo
- To use this algorithm, we run the program with `mpiexec`.
- The first input parameter is the number of instantiated MPI processes `\-n y`, where y is the number of processes.
- The second input parameter is the binary file of the program, with one of four possible optimisations (0,1,2 and 3) in the `tarjan/bin/` folder after compilation, `./tarjan/bin/cuda_mpi_Ox.out` , x representing the chosen optimisation.
- The third input parameter is the graph on which you want to run the algorithm `./tarjan/data/seed.bin`.
- The fourth input parameter is the file that will contain the SCCs found `./tarjan/data/SSC_discovered.bin`.
- An example use in the root directory `./tarjan/` is as follows: `mpiexec \-n 8 ./bin/cuda_mpi_Ox.out ./data/seed.bin ./data/SSC_discovered.bin`.

## Tools

### Generate graphs

**rsg: Random Seed Generator**

- This tool generate a random graph following the graph representation of the `graph.h` library, with max_n_node node and each node have mean number of edge with a variance_edge, the number of nodes follows the Bernoulli distribution.
- To use this tool, we run the program in the `tarjan/bin/` folder after compilation, `./bin/rsg.out`.
- The first parameter is the path of graph generated `./tarjan/data/random-graph.bin`.
- The second parameter is a integer that indicate the number of node of graph.
- The thrid parameter is an integer that indicate the the mean of edge for each node.
- The fourth parameter is the variance of number of edge for each node.

- An example use in the root directory `./tarjan/` is as follows: `./bin/rsg.out ./data/random-graph.bin 1000 15 0.5`.

**rgg: Random Graph Generator**

- This tool generates a graph starting from a seed, representation of the `graph.h` library. This tool uses an interger n to generate a graph with 2^n replicas of the seed keeping all the edges already present in the seed, in addition edges are added between the different seeds of the final graph following the probability passed.
- To use this tool, we run the program in the `tarjan/bin/` folder after compilation, `./bin/rgg.out`.
- The first parameter is the path of seed graph `./tarjan/data/seed.bin`.
- The second parameter is the path of graph generated `./tarjan/data/tile-graph.bin`.
- The thrid parameter is an integer n that indicate the 2^n copy to be created.
- The fourth parameter is the probability of create an edge between a node of a copy and another and viceversa.
- An example use in the root directory `./tarjan/` is as follows: `./bin/rgg.out ./data/seed.bin ./data/tile-graph.bin 2 0.2`.

**sgg: Special Graph Generator**

- This tool generate graph fully connected, graph fully disconnected or graph bipartite, representation of the `graph.h` library.
- The first parameter is the path of graph generated. `./tarjan/data/generated-graph.bin`.
- The second parameter is a integer that indicate the number of node of graph.
- The thrid parameter is an integer that indicate 0 for graph fully disconnected, 1 for graph fully connected, 2 for grapg bipartite.
- An example use in the root directory `./tarjan/` is as follows: `./bin/sgg.out ./data/generated-graph.bin 1`.

## Print Graph

- This tool print to standard output the graph from a file in input.
- To use this tool, we run the program in the `tarjan/bin/` folder after compilation, `./bin/print-graph.out`.
- The first parameter is the path of the graph file to be printed `./data/seed.bin`.
- An example use in the root directory `./tarjan/` is as follows: `./bin/print-graph.out ./data/seed.bin`.

## Print SSC discovered

- This tool print to standard output the SCC discovered from a file.
- To use this tool, we run the program in the `tarjan/bin/` folder after compilation, `./bin/print-SCC.out`.
- The first parameter is the path of the SCC file to be printed `./data/SSC_discovered.bin`.
- An example use in the root directory `./tarjan/` is as follows: `./bin/print-SCC.out ./data/SSC_discovered.bin`.

## Compare

- This tool compare two different SCC discovered file. Checks whether all SSCs in the first file are contained in the second file and vice versa. This is useful to verify the correctness of the parallel algortims by comparing the SSCs found with those found by sequential Tarjan.
- To use this tool, we run the program in the `tarjan/bin/` folder after compilation, `./bin/compare.out`.
- The first parameter is the path of the first SCC file to be compered `./data/SSC_discovered_sequantial.bin`
- The second parameter is the path of the second SCC file to be compered. `./data/SSC_discovered_parallel.bin`
- An example use in the root directory `./tarjan/` is as follows: `./bin/compare.out ./data/SSC_discovered_sequantial.bin ./data/SSC_discovered_parallel.bin`.

### Generate measure

- This tool makes the measurements
- For the use of this tool we have to indicate as the only parameter the algorithm to be used to make the measurements, the possible alternatives: `sequential`, `cuda`, `cuda_texture`, `mpi_cuda` and `sequential_pre`.
- An example use in the root directory `./tarjan/` is as follows: `./measurements/measurement.bash cuda`.

### Extract Graphs and Tables

- To exctract graphs and tables the following pip modules are required:

  sudo apt-get install python3-pip

  pip install numpy scipy pandas matplotlib seaborn prettytable

- Tool that generates tables containing averages of measurements the graphs related to speedup , Amdahl's law and the ideal speedup limit.

- To use such a tool in the extrract.py file, two varaibles `execution_type =` [line 44], `measurement_folder =` [line 45] must be configured

- `execution_type` allows graphs to be generated depending on the algorithm used.

- `measurement_folder` indicates the folder where the measurements are located, in our case only measure and measure_rasperry can be used.

- An example, after setting `execution_type = mpi` and `measurement_folder = measure` use in the directory `./tarjan/measurements/` is as follows: `python3 extract.py`

## Tests

- To execute tests AFTER building the tests with `make tests`, run `make test` (notice the difference between the two).