Reporting Solutions to the 10th DIMACS Implementation Challenge

January 23, 2012

1 General Information

Each (solver, architecture) tuple has to be registered as a submission using EasyChair, using the following link:

`http://www.easychair.org/conferences/?conf=dimacs10`

- In the first step, tick the *Abstract only* box and submit without uploading the solution to get a submission number. This number is used as unique identifier `<solver id>` and has to be included in the name of the solution files. As title use the name of your solver, the abstract should contain a short description of your solver including information about corresponding publications and the hardware used. The submission system requires you to specify at least three keywords. As we do not use this information, you can simply write `keyword1`, `keyword2` and `keyword3`.
- In a second step, for each solver all solutions that shall be considered for the challenge have to be collected in a directory and uploaded in the field "Paper" either zipped (file extension `.zip`) or as a gzipped tarball (file extension `.tgz`), information about the content of this directory is given below. **Please note that in addition to that, for the Pareto challenge you have to run the benchmark test from the last DIMACS challenge to assess the hardware you are using.** The results from that benchmark also have to be included in this directory, for further information consider the last section of this document.

1.1 Deadlines

The deadline for registering solvers (i.e. completing the first step to get a submission number) is February 2nd, 2012, 11:59pm EST. The deadline for submitting solutions (i.e. completing the second step) is February 5th, 2012, 11:59pm EST.

1.2 Several Solvers

If more than one solver shall be considered for the challenge, each solver has to be registered separately.

1.3 Different Hardware Types

If you intend to run a solver on several machines, please register each combination of solver and hardware as a separate submission.

2 Content of `.zip` or `.tgz`-File

Important: Please read the following sections carefully! If you do not follow the guidelines, your solver will not be considered for the challenge.

For each instance two different files have to be submitted, the first one (`.ptn`) containing the actual partition of the vertex set that was computed and the second one reporting meta information like running time and solution quality (`.eval`). If a solution is used for more than one challenge type, then it has to be included multiple times under the appropriate file names.

2.1 Template for file names

File names have to follow the following pattern:

- `<challenge type>#<k>#<solver id>#<category>#<graph name>.ptn`
- `<challenge type>#<k>#<solver id>#<category>#<graph name>.eval`

The wildcards must be replaced in the following way:

- `<challenge type>`:
  - `EC` for Balanced Edge Cut
  - `CV` for Communication Volume
  - `MO` for Modularity
  - `MI` for graph clustering with mixed objective
- `<k>`: Number of parts in solution (4 digits; if less than 1000, fill with preceding zeros) for challenge types `EC` and `CV`, for clustering challenges `MO` and `MI` always use the dummy number `0000`
- `<solver id>`: Corresponds to the unique submission number assigned by EasyChair. This field must contain 3 digits; if the submission number is less than 100, preceding zeros must be used.
- `<category>`: Name of graph category corresponding to the directory names in `http://www.cc.gatech.edu/dimacs10/archive/data/`

- `<graph name>`: Name of the graph file without extension

An example for a file name reporting the solution of solver `38` on the graph `delaunay_n10.graph.bz2` taken from the category `delaunay` that partitions the vertex set in 64 parts and is intended to be used for the challenge type `EC` is

`EC#0064#038#delaunay#delaunay_n10.ptn`

2.2 Format of `.ptn` and `.eval` files

- `.ptn`: The partition/cluster assignment files have n lines. In line i the assignment value for vertex i is given as an integer between 0 and p-1, where p is the number of partitions.
- `.eval`:
  - Line 1: Quality of the solution according to the corresponding challenge type. For `EC` and `CV`, these are integers that have to be given in full precision. For `MO`, this is a rational number that has to be correct to nine decimal places. For `MI`, this is a comma-separated list of 6 values in the following order:

    `<Minimum intra-cluster density>, <Average isolated inter-cluster conductance>, <Average isolated inter-cluster expansion>, <Performance>, <Coverage>, <Mirror Coverage>`

    Each of these values has to be correct to nine decimal places.

    The following lines are necessary for the Pareto challenge only. They can be left out if you do not want to participate in it.
  - Line 2: Running time the solver needed to compute the solution (in seconds, correct to three decimal places). Use a monotonic wall-clock that starts after reading the DIMACS graph file into the user's data format and stops before writing the `.ptn` file. The evaluation for line 1 is untimed.
  - Line 3: CPU name
  - Line 4: Number of CPU cores (`p`), number of GPUs (or other accelerator) (`g`), number of sockets/paths to memory (`m`) used to compute the solution as three comma-separated values `p, g, m`.
  - Line 5: Name of accelerator if one was used, i. e., if `g` > 0 in the previous line.

2.3 Results from Benchmark

The benchmark can be downloaded from the homepage of the 9th DIMACS Implementation Challenge:

`http://www.dis.uniroma1.it/~challenge9/download.shtml`

Please follow the instructions in `README.txt` and insert `#include <cstdlib>` into `merger.cpp` if necessary. After running the benchmark test on your machine, copy the folder `results` to the root of the directory you are uploading. The resulting files will contain the average time per operation on a number of benchmark problems. The capability of your system will be measured based on this benchmark and the number of processing elements you use in your challenge computations.

When benchmarking your system with the code from the previous DIMACS challenge, we expect you to use a high level of compiler optimization. Use the Makefile as is if possible. If you do not use GCC, use an optimization level that is similar to the one in the Makefile used for GCC. As a general rule, your production code for the challenge should not have better compiler optimization flags than the benchmark code. **If in doubt, please contact us!**