CPPR Manual

1 Description

cppr is an implementation of the colored parallel push-relabel algorithm. There is also a sequential fifo push-relabel algorithm implemented mostly using same routines. Two different graph representations are implemented, adjacency list and star graph representations.

This code is structured as a library but it also contains a standalone tool to use as a solver for DIMACS maximum flow problem files. Solution is then given as a DIMACS maximum flow solution file with optional flow value assignments. See the provided samples as an example for these file formats.

This documentation assumes a UNIX based operating system with essential build tools installed and a POSIX compatible shell for commands. You may need to adjust these to your setup for other combinations.

This same content can be read from both `README` and `doc/manual.pdf` files.

2 Building

Run `make` to compile the tool.

Run `make run` to run with the sample input.

Run `make test` to compile and run unit tests.

See `Makefile` for compile time options and their default values.

3 Installation

If you want to install the standalone tool, you can simply copy the binary to a directory in `PATH` variable:

```
cp bin/cppr /usr/local/bin
```

If you want to install the library, you can copy header files in `include` directory to a system include directory:

```
mkdir -p /usr/local/include/cppr
cp include/* /usr/local/include/cppr
```

You can then include these files with the corresponding prefix in your source code:

```
...
#include <cppr/dimacs.hpp>
#include <cppr/push_relabel.hpp>
...
```

You may need to elevate permission to use these commands (e.g. `sudo`).

4 Compiler

You need a compiler with at least C++11 and OpenMP 3.1 support for compilation. This roughly corresponds to version 4.8 onwards for GCC and version 3.9 onwards for Clang. Detailed feature support information for GCC and Clang can be found in following pages if you need to use older versions:

- `https://gcc.gnu.org/projects/cxx-status.html`
- `https://gcc.gnu.org/wiki/openmp`
- `https://clang.llvm.org/cxx_status.html`
- `https://openmp.llvm.org/`

Note: Although Clang 3.8 supports the necessary specifications, there is a bug crashing the frontend when references are used in OpenMP reduction clauses which seems to be fixed in 3.9 onwards.

5 Usage

cppr tool can read the input either from a file given as the first argument or from stdin when an argument is not provided. Output is given to stdout by default which can instead be written to a file either using `-o` option with a file name or using shell redirections.

An example DIMACS input file and the corresponding DIMACS output file are provided in `test/sample` directory. DIMACS file format for the maximum flow problem is described here:

- `http://lpsolve.sourceforge.net/5.5/DIMACS_maxf.htm`

The example input file is as follows:

```
$ cat test/sample/inp.max
c This is a simple example file to demonstrate the DIMACS
c input file format for maximum flow problems. The solution
c vector is [5,10,5,0,5,5,10,5] with cost at 15.
c Problem line (nodes, links)
p max 6 8
c source
n 1 s
c sink
n 6 t
c Arc descriptor lines (from, to, capacity)
a 1 2 5
a 1 3 15
a 2 4 5
a 2 5 5
a 3 4 5
a 3 5 5
a 4 6 15
a 5 6 5
c
c End of file
```

This can be run with cppr tool as follows:

```
$ bin/cppr test/sample/inp.max
c
c cppr v1.0.0
```

```
c
c # nodes : 6
c # arcs  : 8
c
c graph     : star
c problem   : maximum flow
c algorithm : colored parallel push-relabel (8 threads)
c
c # pushes          : 7 + 1 = 8
c # relabels        : 1 + 0 = 1
c # discharges      : 1 + 0 = 1
c # global relabels : 1 + 1 = 2
c # colors          : 2
c # color ticks     : 2 + 1 = 3
c
c solution is not checked
c
s 15
c
c total measured time           : 0.000802795
c - init time                   : 4.0107e-05
c   - read time                 : 3.7393e-05
c   - convert residual time     : 2.714e-06
c - solve time                  : 0.000747983 + 1.4705e-05 = 0.000762688
c   - color graph time          : 4.432e-06
c   - discharge time (par)      : 3.8084e-05 + 4.134e-06 = 4.2218e-05
c   - global relabel time (par) : 0.000705467 + 1.0571e-05 = 0.000716038
c - report time                 : 0
c   - check solution time       : 0
c   - write time                : 0
```

There are also some small examples generated with the tools used in DIMACS challenge in `test/dimacs` directory. Output from cppr tool for these examples can be found in the same directory.

Run `bin/cppr -h` to see all runtime options along with the usage.

6 OpenMP Options

You can use `OMP_NUM_THREADS` variable to set the number of threads used for the parallel algorithm:

```
OMP_NUM_THREADS=2 bin/cppr test/sample/inp.max
```

You can use `OMP_PROC_BIND` variable to set processor affinities to prevent thread migrations (recommended):

```
OMP_PROC_BIND='TRUE' bin/cppr test/sample/inp.max
```

When these variables are not set, default values of these options are implementation dependent.

7 Intel TBB Allocator

You can replace the standard memory allocator with scalable memory allocator from Intel TBB library to avoid false sharing (recommended). You can either do this at compile time by setting `LDLIBS` while compiling:

```
LDLIBS=-ltbbmalloc_proxy make && bin/cppr test/sample/inp.max
```

Or at load time using `LD_PRELOAD` trick:

```
make && LD_PRELOAD=libtbbmalloc_proxy.so.2 bin/cppr test/sample/inp.max
```

If TBB is installed in a non-standard location, you may need to add `-ltbbmalloc` to `LDLIBS` and the path to TBB library to `LIBRARY_PATH` and `LD_LIBRARY_PATH` beforehand:

```
LIBRARY_PATH="/path/to/tbb:$LIBRARY_PATH" \
LDLIBS='-ltbbmalloc -ltbbmalloc_proxy' \
make && \
LD_LIBRARY_PATH="/path/to/tbb:$LD_LIBRARY_PATH" \
bin/cppr test/sample/inp.max
```

Or if you want to use the second method, then you only need to add the path to `LD_LIBRARY_PATH` before running:

```
make && \
LD_LIBRARY_PATH="/path/to/tbb:$LD_LIBRARY_PATH" \
LD_PRELOAD=libtbbmalloc_proxy.so.2 \
bin/cppr test/sample/inp.max
```

See the related section in TBB documentation for more information:

- `https://software.intel.com/en-us/node/506096`

8 Files

The code is structured as a header only C++ library and also includes a standalone tool. If you want to use it as a library, you only need to include appropriate header files in your source. You can use the standalone tool as an example for this purpose. Also, if you create a new source file under tool directory, it will automatically be compiled as a separate tool.

Brief descriptions of subdirectories are as follows:

```
.
|-- bin/                           -- tool binaries (after building)
|   `-- cppr*
|-- build/                         -- build files (after building)
|   |-- dep/                       -- dependency files (after building)
|   |   |-- color_queue_test.d
|   |   |-- cppr.d
|   |   |-- multi_queue_test.d
|   |   `-- push_relabel_test.d
|   `-- test/                      -- test binaries (after building)
|       |-- color_queue_test*
|       |-- multi_queue_test*
|       `-- push_relabel_test*
|-- doc                            -- documentation
|   |-- manual.pdf
|   `-- manual.tex
|-- include/                       -- header files
|   |-- color_queue.hpp
|   |-- common.hpp
|   |-- dimacs.hpp
|   |-- greedy_coloring.hpp
|   |-- list_graph.hpp
|   |-- multi_queue.hpp
|   |-- push_relabel.hpp
|   `-- star_graph.hpp
|-- Makefile
|-- README
|-- test/                          -- source files for tests
|   |-- color_queue_test.cpp
|   |-- multi_queue_test.cpp
|   |-- push_relabel_test.cpp
|   `-- sample/                    -- sample DIMACS input/solution
|       |-- inp.max
|       `-- sol.max
`-- tool/                          -- source files for tools
```

```
    `-- cppr.cpp

9 directories, 26 files
```