# 1 ARTIFACT DESCRIPTION: ENABLING HIGH-LEVEL GRAPH PROCESSING VIA DYNAMIC TASKING

## 1.1 Abstract

This description contains the information needed to launch some experiments of the SC18 poster "Enabling High-Level Task Processing via Dynamic Tasking". In particular, we explain how to: (1) compile the proposed variant of the GMT framework; (2) compile the SHAD library and link it to the GMT variant; (3) run the SHAD application for triangle counting.

## 1.2 Description

*1.2.1 Check-list (artifact meta information).*

- **Algorithm:** Triangle Counting
- **Program:** C and C++ libraries
- **Compilation:** g++ (GCC) 7.1.0 with the -O3 flag
- **Data set:** Erdös-Rényi Graphs[1] with scale factors from 23 to 25
- **Output:** Time to solution
- **Experiment workflow:** clone GMT, build GMT, install GMT, clone SHAD, build SHAD, obtain the datasets, run the binaries
- **Experiment customization:** number of GMT processes
- **Publicly available?:** yes

*1.2.2 How software can be obtained.* The proposed GMT variant can be cloned from GitHub at the following URL: https://github.com/droccom/gmt. SHAD can be cloned from GitHub at the following URL: https://github.com/pnnl/shad. The source code for the triangle counting is in the file: `examples/edge_index_graph/ei-triangle-count.cc`.

*1.2.3 Hardware dependencies.* A homogeneous cluster is needed to execute the experiments in a distributed-memory setting.

*1.2.4 Software dependencies.* SHAD requires CMake 3.1 or above and a C++14 compliant compiler (e.g., g++ 5 and above). The same setup can be used to compile GMT as well. For better performance, the tcmalloc concurrent allocator (from the gperftools suite) should be used. An MPI implementation (e.g., OpenMPI) is needed to execute the experiments in a distributed-memory setting.

*1.2.5 Datasets.* The graphs can be obtained at the following URL: https://www.cc.gatech.edu/dimacs10/archive/er.shtml.

## 1.3 Installation

Navigate to your sandbox root:

```
cd /path/to/sandbox/root
```

Clone, build, and install the GMT variant at the location stored in the environment variable `GMT_ROOT`:

```
export GMT_ROOT=/path/to/gmt/root
git clone https://github.com/droccom/gmt.git
cd gmt
mkdir build && cd build
cmake .. -DCMAKE_INSTALL_PREFIX=$GMT_ROOT \
         -DCMAKE_BUILD_TYPE=Release
make && make install
```

Clone and build SHAD, possibly linking with the tcmalloc allocator from the gperftools suite rooted at `GPERFTOOLS_ROOT`:

```
cd /your/sandbox/root
git clone https://github.com/pnnl/shad.git
cd shad
mkdir build && cd build
cmake .. -DCMAKE_BUILD_TYPE=Release   \
         -DSHAD_ENABLE_UNIT_TEST=OFF  \
         -DSHAD_RUNTIME_SYSTEM=GMT    \
         -DGMT_ROOT=$GMT_ROOT         \
         -DGPERFTOOLS_ROOT=$GPERFTOOLS_ROOT
make -j
```

Save the location of the triangle-counting binary:

```
export APP_BIN=/path/to/sandbox/root\
/shad/build/examples/edge_index_graph
```

## 1.4 Experiment workflow

Obtain the graph datasets (about 10 GB in total):

```
cd /path/to/sandbox/root
mkdir datasets && cd datasets
for scale in 23 24 25; do
wget https://www.cc.gatech.edu/dimacs10/archive\
/data/er/er-fact1.5-scale$scale.graph.bz2
bunzip2 er-fact1.5-scale$scale.graph.bz2
done
```

Run the experiment on the scale-23 graph, over two nodes of the cluster:

```
mpirun -np 2 $APP_BIN er-fact1.5-scale23.graph
```

## 1.5 Evaluation and expected result

The numbers reported in the poster were observed on a 16-nodes cluster, equipped with two Intel Xeon E5-2680 v2 CPUs, working at 2.8 GHz (hyper-threading disabled) and 768 GB of memory per node. We obtained the best performance with a per-socket mapping, thus launching 32 GMT processes in total:

```
for scale in 23 24 25; do
mpirun -np 32 -npernode 2 --bind-to socket $APP_BIN \
er-fact1.5-scale$scale.graph
done
```

## 1.6 Notes

Depending on the platform, the syntax for launching the experiments may vary. For instance, the syntax for specifying the number of processes depends on the MPI implementation. Moreover, it could be necessary to interact with a resource allocator on the cluster (e.g., SLURM) by wrapping the reported code into specific commands (e.g., `salloc`/`sbatch`).

[1] https://www.cc.gatech.edu/dimacs10/archive/er.shtml