1. Artifact Description: [ Generating Families of Practical Matrix Multiplication Algorithms ]

1.1 Abstract
The artifact contains partial components of the implementations of IPDPS'17 paper “Generating Families of Practical Matrix Multiplication Algorithms” [1].

1.2 Description
1.2.1 Check-list (artifact meta information)
- **Program:** C99, python, AVX inline assembly
- **Compilation:** python2.7; icc/icpc version 15.0.3 with the gcc 4.9.3 Standard Template Library for sequential and parallel experiment.
- **Hardware:** Maverick supercomputer at TACC.
- **Output:** Effective GFLOPS performance, as shown in Figure 5.
- **Experiment workflow:** Decompress the source code, configure and compile the code, check the dataset and environment, and execute the code.
- **Publicly available?:** Yes.

1.2.2 How software can be obtained (if available)
The source code can be obtained in the URL below:
http://www.cs.utexas.edu/users/jianyu/code/fmm-gen.tar.gz

1.2.3 Hardware dependencies
x86-64 processor that supports AVX instructions.

1.2.4 Software dependencies
Linux or macOS/OS X, autotools, Intel icc/icpc compiler 15 or later and compatible GCC 4.9 or later, OpenMP environment, SLURM queuing system.

1.3 Installation
Follow the build instructions in README.md, found in the source code top folder after extracting the fmm-gen.tar.gz from the URL above.

1.4 Experiment workflow
- Download fmm-gen.tar.gz from the URL above.
- Decompress the source code:
  ```
  $ tar -zxvf fmm-gen.tar.gz
  $ cd fmm-gen/meta
  ```
- Set up environment variables: Replace $core_num with the number of cores the user wants to run.
  ```
  $ export OMP_NUM_THREADS=$core_num
  $ export KMP_AFFINITY=compact,1
  ```
- Code generators:
  ```
  $ python control.py $\{N\} $m1n1p1 $L1 \n  $m2n2p2 $L2 \ldots \n  $m{N}n{N}p{N} $L{N} \n  $\{pack_type\} $\{gen_path\}
  ```
e.g.
  ```
  $ python control.py 2 222 1 323 1 abc \n  ${HOME}/fmm-gen
  $ python control.py 1 222 2 abc ...
  ```
  This script will generate the code and compile it.
  To further execute the code, go to the generated code directory (e.g. $HOME/fmm-gen/222-1_333-1_abc, or ../222-2_abc).
  When $\{core_num\}$ is equal to 1, run
  ```
  ./test/test_xxx-x_st.x $m $n $k
  ```
  When $\{core_num\}$ is greater than 1, run
  ```
  ./test/test_xxx-x_mt.x $m $n $k
  ```
- If you have access of a job submission system on a cluster, change the path_prefix variable in config.py, then:
  ```
  $ python run_sbatch_script.py
  ```
  This script will generate the code for all implementations, compile them, and submit the jobs to SLURM submission queue for execution.
- Hybrid partitions:
  ```
  $ python control.py 1 222 1 abc
  $ python control.py 1 222 2 abc
  $ python control.py 1 232 1 abc
  $ python control.py 1 232 2 abc
  $ python control.py 1 333 1 abc
  $ python control.py 1 333 2 abc
  $ python control.py 2 222 1 232 1 abc
  $ python control.py 2 222 1 333 1 abc
  ```
- Model:
  ```
  $ python model_gen.py
  ```
  This script will generate csv files for plotting the modeled performance curves.

1.5 Evaluation and expected result
The output will include the following components:
- **Input problem size:**
• Running time (in seconds).
• Effective GFLOPS (in Figure 5).

The user can compare the relative Effective GFLOPS for different implementations. The trend should match the performance curves shown in this paper. Since the machines may be different from ours, the absolute GFLOPS could be different.

References