Appendix: Artifact Description/Artifact Evaluation

ARTIFACT IDENTIFICATION

We presents a holistic approach–PrecTuner–by closely coupling the code generator and the autotuner via only one parameter r . Initialized by automatically sampled values, r is first used to generate several code variants in the polyhedral model, combining this optimization with loop transformations. These code variants are then used to solve a performance model expressed in terms of r , possibly under a quality degradation budget. The value of r that produces the best-performing mixed-precision code is finally predicted without evaluating all code variants.

As for the computational artifact of this work(https://github.com/sheenisme/lnlamp.git), it implements automatic code generation based on PPCG(http://ppcg.gforge.inria.fr/), and on Python and Shell scripts for automatic tuning, which closely coupling the code generator and the autotuner via only one parameter r .

The usage of PrecTuner is as follows:

```
Usage: lnlamp file [options] ...
Options:
 -s|--specify-output specify the target file
 -r|--rate-array     specify the rate-array of profiling,
    (default is:0 12 25 37 50 62 75 87 99)
 -e|--error-threshold specify the error-threshold
    (default is:999......)
 -v|--version    lnlamp version
 -t|--tile       lnlamp specify the tile-sizes
    (default none) in PPCG
 -o|--openmp     lnlamp specify uses the --openmp in PPCG,
    and multi-threads(default is:4) running
 -a|--sched-algo lnlamp specify the schedule-algorithm
    to isl|feautrier|no in PPCG, default is isl
 -i|--isl-flags  lnlamp specify the flags in isl library
    of PPCG, default is null
Examples:
lnlamp hello.c -s hello_lnlamp.c
Template:
lnlamp <source-file> [Options] <target-file>
```

It is worth mentioning that all the experimental data in this article were obtained using this tool. In order to better test our method, we embedded the experimental test sub-module (polybench_benchmark) in this tool and encapsulated the testing process into a corresponding script so that our peers can quickly reproduce our experimental process.

REPRODUCIBILITY OF EXPERIMENTS

0.1 Getting Started Guide

Note: The default installation path for all tools in this version of artifact is under '/home/sheen' and there is hard code for this path in the script code, we recommend soft linking your custom folder to '/home/sheen' so you can use it quickly and correctly. We also recommend that all users try installing with a dockerfile first, so that even if you have to install locally, you can refer to the docker image for help with some issues.

System configuration:

```
Cpu   : Intel i5 processor (recommended)
Gpu   : NVIDIA RTX 2070    (recommended)
Os    : Ubuntu 20.04.5 LTS (recommended)
Gcc   : 9.4.0  (recommended)
Clang : 12.0.1 (recommended)
Memory: Recommended 32G and above
Disk  : Greater than 256G (artifact need about 124G)
```

step1 install PrecTuner

```
# prepare
sudo apt update && sudo apt upgrade -y
sudo apt-get install gcc g++ git vim make bc python python3-pip
sudo apt install automake autoconf libtool pkg-config
     libgmp3-dev libyaml-dev libclang-dev llvm clang
pip install pandas numpy matplotlib
cd /home/sheen/
mkdir lnlamp-install
git clone https://github.com/sheenisme/lnlamp.git
cd lnlamp/
./get_submodules.sh
./autogen.sh
./configure --prefix=/home/sheen/lnlamp-install
make
make install
```

For further installation help please refer to: https://repo.or.cz/ppcg.git.

step2 install LLVM

```
# prepare
sudo apt install cmake
cd /home/sheen/
git clone https://github.com/sheenisme/llvm-project.git
cd llvm-project/
mkdir llvm-install
git checkout origin/release/12.x
cmake -S ./llvm -B llvm-build -G Ninja
 -DCMAKE_BUILD_TYPE="Release" -DLLVM_VERSION_MAJOR="12"
 -DLLVM_TARGETS_TO_BUILD="X86" -DLLVM_ENABLE_PROJECTS=
 "clang;clang-tools-extra;compiler-rt;openmp;polly"
 -DLLVM_BUILD_LLVM_DYLIB=ON -DLLVM_LINK_LLVM_DYLIB=ON
 -DCMAKE_INSTALL_PREFIX=/home/sheen/llvm-project/llvm-install
cd llvm-build/
cmake --build .
```

Note: this process requires a lot of hard drive space, so make sure you have plenty of space.

For further installation help please refer to: https://github.com/llvm/llvm-project.

step3 install LuIs

```
cd /home/sheen/
git clone https://github.com/sheenisme/TAFFO.git
cd TAFFO/
mkdir taffo-install
export LLVM_DIR=/home/sheen/llvm-project/llvm-install
cmake -S . -B build -DTAFFO_BUILD_ORTOOLS=ON
 -DCMAKE_INSTALL_PREFIX=/home/sheen/TAFFO/taffo-install
cd build/
cmake --build .
```

Note: This process requires downloading a number of dependencies from github. If you have a poor internet connection or a timeout error, please run the `cmake -S . -B build -DTAFFO_BUILD_ORTOOLS=ON -DCMAKE_INSTALL_PREFIX=/home/sheen/TAFFO/taffo-install` command several times until it works.

For further installation help please refer to: https://github.com/TAFFO-org/TAFFO.

step4 install Pluto

```
# prepare
sudo apt-get install flex bison texinfo
cd /home/sheen/
git clone https://github.com/sheenisme/pluto.git
cd pluto/
mkdir pluto-install
git submodule init
git submodule update
./autogen.sh
./configure --prefix=/home/sheen/pluto/pluto-install
make
make install
```

For further installation help please refer to: https://github.com/bondhugula/pluto.

step5 Configuring environment

```
export ROOT_INSTALL_DIR=/home/sheen/
# setting pluto
export PATH=${ROOT_INSTALL_DIR}/pluto/pluto-install/bin:$PATH
# setting LLVM
export LLVM_DIR=${ROOT_INSTALL_DIR}/llvm-project/llvm-install
export PATH=${LLVM_DIR}/bin:$PATH
export LD_LIBRARY_PATH=${ROOT_INSTALL_DIR}/lib:$LD_LIBRARY_PATH
# setting TAFFO
export PATH=${ROOT_INSTALL_DIR}/TAFFO/taffo-install/bin:$PATH
export LD_LIBRARY_PATH=
${ROOT_INSTALL_DIR}/TAFFO/taffo-install/lib:$LD_LIBRARY_PATH
# setting prectuner
export PATH=${ROOT_INSTALL_DIR}/lnlamp-install/bin:$PATH
export LD_LIBRARY_PATH=
${ROOT_INSTALL_DIR}/lnlamp-install/lib:$LD_LIBRARY_PATH
# setting polybench
export CPATH=/usr/local/include:
${ROOT_INSTALL_DIR}/lnlamp/polybench_benchmark/utilities:$CPATH
```

You will also need to execute the above shell command (or using `direnv` tool, or run ‘`sh /home/sheen/lnlamp/.envrc`‘) to set the relevant environment variables.

Now all tools are installed.

0.2 Reproducing Results

1. Reproducing the Effects of Performance Prediction.
The prediction results can be reproduced by executing the following command:

```
cd /home/sheen/lnlamp/polybench_benchmark/
nohup ./lnlamp_tests_for_judgement.sh &

cp nohup.out ./plot_scripts/
cp ./scripts/benchmark_result_perf-Reliable.log
   ./plot_scripts/
cd ./plot_scripts/
sh get_perf_gain.sh
sh get_predict.sh
```

The `lnlamp_tests_for_judgement.sh` command will generate files: `nohup.out`, `scripts/benchmark_result_perf-Reliable.log`, `scripts/benchmark_test.log`, and `scripts/benchmark_mean_result.log`. The `nohup.out` contains the execution log of the `PrecTuner` and its prediction results; `scripts/benchmark_mean_result.log` and `scripts/benchmark_test.log` is the log of the empirically executed; `scripts/benchmark_result_perf-Reliable.log` is the empirically executed results automatically collated by the script.

After all the above commands are executed, two files `result_real.csv` and `result_predict.csv` are generated, where the data in the former is the performance gain when the test case is actually run, and the latter is the performance gain curve predicted by prectuner. Plotting the former and the latter in the same axis yields **Figure 10**.

Note: The `lnlamp_tests_for_judgement.sh` script execution time for this experiment may be more than **12 hours**.

2. Reproducing the Ablation Study of the Optimizations.
The results can be reproduced by executing the following command:

```
cd /home/sheen/lnlamp/polybench_benchmark/
nohup ./lnlamp_tests_for_performance.sh &
```

These commands will generate folders: `only-mix`, `schedule`, `schedule_mix`, `schedule_tile`, `schedule_tile_mix`. However, as the execution results are stored, these `vra` folders may take up a large amount of hard disk storage space(A folder of that requires 10G), so please delete these runtime data in time to ensure that your machine has sufficient storage space.

The results of the respective executions are recorded in `vra.txt` for each of these folders. The first column of letters in these vra.txt files represents the name of the test case and the last column of numbers represents the performance speedup. **Figure 11** can be easily obtained using these speedup data.

Note: The script execution time for this experiment was approximately **10 hours**.

3. Reproducing the Compatibility with ErrorBudgets.
The result for an error threshold of `0.1` can be obtained with the following command:

```
cd /home/sheen/lnlamp/polybench_benchmark/
nohup ./lnlamp_tests_for_performance_err-thr.sh &
```

These commands will generate folders: `temp_test_res_-e-1`, `temp_test_res_-e-0.1`, `temp_test_res_-e-0.01`, `.....` , `temp_test_res_-e-0.000001`. It also will generate file `-nohup.out`, which contains the execution log and results of runs with these error threshold.

Figure 13 is drawn using the data from the `vra.txt` of these folders and `nohup.out`.

Note: The script execution time for this experiment was approximately **2 hours**.

4. Reproducing the Comparison with the State of the Art.
(1) Reproducing the PrecTuner performance. The test in this section is the same as **Reproducing the Ablation Study of the Optimizations**, so the corresponding data can be obtained directly from the latter.
(2) Reproducing the LuIs performance. The results can be reproduced by executing the following command:

```
cd /home/sheen/TAFFO/test/polybench-c-4.2.1-beta/
export LLVM_DIR=/home/sheen/llvm-project/llvm-install
nohup ./collect-fe-stats.sh luis_test_res &
```

These commands will generate folder-`luis_test_res` in order to obtain runtime result data(error and performance acceleration),and generate file `-nohup.out` ,which contains the execution log.

The `LuIs` performance speedup in the last column in `luis_test_res/vra.txt`. It should be noted that the Relative error and Absolute error data of `LuIs` is in the file - `luis_test_res/vra.txt`.
(3) Reproducing the Pluto performance. The results can be reproduced by executing the following command:

```
cd /home/sheen/pluto/polybench_benchmark
nohup ./pluto_part_test.sh &
```

These commands will generate folder-`pluto_test_result` in order to obtain runtime result data(error and performance acceleration), and generate file -`nohup.out` ,which contains the execution log.

The `Pluto` performance speedup in the last column in `pluto_test_result/vra.txt`.

So, Figure 12 can be drawn using the above performance speedup data. Meanwhile the error data in table2 are also available by the above error data(`e_abs` means absolute error, `e_perc` means relative error).

Note: The execution time of the scripts corresponding to each part of this experiment was approximately **3 hours**.

5. Reproducing the Scalability to Parallel Execution.
(1) Reproducing the PrecTuner parallel execution. The results can be reproduced by executing the following command:

```
cd /home/sheen/lnlamp/polybench_benchmark/
nohup ./lnlamp_tests_for_performance_omp.sh &
```

These commands will generate folders: `omp_test_res_-o_2`, `omp_test_res_-o_4`, `omp_test_res_-o_8`. The results of the respective executions are recorded in `vra.txt` for each of these folders.
(2) Reproducing the Pluto parallel execution. The results can be reproduced by executing the following command:

```
cd /home/sheen/pluto/polybench_benchmark
nohup ./pluto_part_test_omp.sh &
```

These commands will generate folders: `pluto_test_result_2`, `pluto_test_result_4`, `pluto_test_result_8`. The results of the respective executions are recorded in `vra.txt` for each of these folders.

So, Figure 14 can be drawn using these data.

Note: The execution time of the scripts corresponding to each part of this experiment was approximately **8 hours**.
(3) Reproducing the CUDA code execution. The results can be reproduced by executing the following command:

```
cd /home/sheen/lnlamp
git apply cuda_test_update.patch
make
make install
cd polybench_benchmark
git apply cuda_test.patch
cd utilities
perl clean.pl ../
perl makefile-gen.pl ../ -cfg
cd ..
nohup ./lnlamp_tests_for_cuda.sh &
```

These commands will generate file -`nohup.out` ,which contains the execution timing result.**Figure 15 can be drawn using these data.**

Please note in particular that the success of the experiments in this section assumes that you have installed the GPU driver and the NVCC compiler. The currently known adapted version of the nvcc compiler is NVCC 11.1.

Note: The script execution time for this experiment was approximately **4 hours**.

6. Reproducing the performance of a Real-life Application.
Reproducing the LU execution. The results can be reproduced by executing the following command:

```
cd /home/sheen
git clone https://github.com/sheenisme/SNU_NPB.git
cd SNU_NPB/NPB3.3-SER-C/LU
nohup ./run_some_rate.sh &
```

These commands will generate file -`nohup.out` ,which contains the execution timing result.

Note: The script execution time for this experiment was approximately **2 hours**.

7. Automatic generation and cleaning of configuration information.
In addition, there are a number of `perl` scripts for automated testing in the `/home/sheen/lnlamp/polybench_benchmark/utilities` ,which can be executed to achieve the appropriate functionality with the following commands:

```
perl header-gen.pl ../     #generates header files.
perl makefile-gen.pl ../ -cfg     #generates make files.
perl clean.pl ../ #runs make clean and removes Makefile.
```

Note: The script execution time for this experiment was approximately **2 minutes**.

8. Some of the problems that may be encountered.

- If `prectuner` does not output any results or hints, then please use the `cd` command to go to the root directory where the source code is located and re-execute the command. If the same problem still occurs, then please follow the error hints in the log file generated in the source code directory (for example: `lnlamp_internal_usage.py.log`) to solve the corresponding errors. And, in general, you only need to install the corresponding package or tool according to the prompt.
- If you encountered `lnlamp: error: PPCG Codegen meets errors` errors, Then please execute the `PPCG CMD` command immediately after it and solve the corresponding problem according to the error or prompt message of the command. Generally, the command takes the form of `ppcg --target c --no-automatic-mixed-precision <input>.c` or `ppcg --target c -R 50 <input>.c`
- If you encountered `fatal error: 'polybench.h' file not found` errors, then execute the `export CPATH=/home/sheen/lnlamp/polybench_benchmark/utilities:$CPATH` command can solve this error.
- If some error(such as 124 or 127) were encountered when executing `heat-3d` by `pluto` , perhaps you can resolve any problems you may encounter by executing the following command:

```
# compiler again
/home/sheen/llvm-project/llvm-install/bin/clang -I
 ./stencils/heat-3d -I./utilities -I./ -DPOLYBENCH_TIME
 -DPOLYBENCH_DUMP_ARRAYS -DPOLYBENCH_STACK_ARRAYS
 -DCONF_GOOD -DLARGE_DATASET -lm -O3 build/
 heat-3d.out.1.taffotmp.ll -o build/heat-3d.pluto.out

# run and get result
./taffo_run.sh --times=20
./taffo_validate.py > result.txt
```

0.3 Other Notes

In addition, we provide a `dockerfile`(https://github.com/sheenisme/lnlamp/blob/master/Dockerfile) file for quick installation, but this installation is not recommended given the instability of performance testing in virtual machines.

It is important to note that we **highly recommend using the Dockerfile to install and use our environment**, even if you want to experiment on a local machine(the operating system we recommend is Ubuntu 20.04.5 LTS - Linux 5.15.0-67-generic) to get stable performance data, the installation steps are recommended to follow the steps described in the dockerfile, and if you want to use a custom path, it is currently recommended to use a soft link(‘`ln -s /your_dir /home/sheen`‘) for the purpose.