A. Artifact Appendix

A.1 Abstract

The artifacts in this paper are in the form of standalone Python scripts for the experiments. Our experiments are primarily tested on GCC 9.2.0, and different compiler versions and hardware may cause the results to vary. Due to the potential for performance variability across different environments, particularly in VM images or Docker, where virtualization layers may introduce additional scheduling overhead, I/O latency, or performance jitter, we recommend running the experiments on a bare-metal machine. All necessary materials, including the source code, scripts, and detailed documentation, are provided via `https://github.com/PKU-ASAL/GroupTuner`, enabling users to install dependencies and reproduce results in a native environment.

A.2 Badging Statement

We seek to obtain the following ACM Artifact Evaluation badges for our submission: **Avaliability(green), Reproducibility(blue) and Functionality/Resuability(red).**

A.3 Artifact check-list (meta-information)

- **Algorithm: GroupTuner**
- **Program: Programs select from cBench and polyBench.**
- **Compilation: Does not need to be compiled, exists as python scripts that can be run directly.**
- **Transformations: GCC will apply transformation based on the customized option selections.**
- **Binary: Original code is included.**
- **Data set: cBench and polyBench.**
- **Run-time environment: Ubuntu 20.04.6 LTS on a bare-metal machine**
- **Hardware: Ubuntu server equipped with an 11th Gen Intel(R) Core(TM) i7-11700 @ 2.50GHz (8 cores) and 16GB memory.**
- **Run-time state: Disable the Turbo Boost feature of Intel, run only one program at a time on the entire server, binding the execution process of program to a fixed CPU core.**
- **Execution: Directly run python scripts.**
- **Metrics: Execution time improvement over -O3.**
- **Output: Graphs and Tables of the overall performance comparsion. The corresponding scripts of figures and tables are included in the Github.**
- **Experiments: Follow the README and scripts in the Github.**
- **How much disk space required (approximately)?: The GroupTuner code is ablout 1MB, the dataset is about 35M and the generated experiment result is about 1GB.**
- **How much time is needed to prepare workflow (approximately)?: It takes 2 hours to download and install GCC and relative python packages.**
- **How much time is needed to complete experiments (approximately)?: It takes about over 1 month to complete all experiments. To facilitate evaluation, we provide a subset of experiments that approximately takes about 3 days.**
- **Publicly available?: Yes.**
- **Code licenses (if publicly available)?: MIT License.**
- **Data licenses (if publicly available)?: MIT License.**
- **Workflow automation framework used?: Python scripts.**
- **Archived (provide DOI)?: No.**

A.4 Description

A.4.1 How to access

Download the artifact from `https://github.com/PKU-ASAL/GroupTuner`.

A.4.2 Hardware dependencies

Ubuntu server equipped with 8 11th Gen Intel(R) Core(TM) i7-11700 @ 2.50GHz and 16GB memeory.

A.4.3 Software dependencies

Ubuntu 20.04.6 LTS, GCC 9.2.0, perf 5.13.19, conda 24.9.2 and Python packages listed in requirements.txt.

A.4.4 Data sets

cBench and polyBench which are included in the Dataset/dataset.

A.5 Installation

To ensure the stability of the experimental results, we disable Turbo Boost and bind each process to a set of isolated CPU cores. And we use Perf to measure the execution time of program accurately. The binding is automatically handled by our execution scripts, and reviewers only need to disable Turbo Boost and isolate a specific CPU core manually.

- **Disable the Turbo Boost feature of Intel. Reboot the server and enter the BOIS setup. Locate the CPU-related settings, typically found under sections such as *Performance* or *Advanced*. Then look for the *Intel Turbo Boost* option and set it to *Disabled* or *Off*. After making the changes, save the settings and restart the server.**
- **Isolate a specific CPU core (core 0 in our experiment).**

```
sudo vim /etc/default/grub
# Add isolcpus=0 to the GRUB_CMDLINE_LINUX
    configuration line.
sudo update-grub
sudo reboot
```

- **Install Perf.**

```
sudo apt update
sudo apt install linux-tools-common linux-tools-$(
    uname -r) linux-tools-generic
sudo sh -c 'echo␣-1␣>␣/proc/sys/kernel/
    perf_event_paranoid'
```

- **Install GCC-9.2.0 using the script we provided, and GCC will be installed in GroupTuner/gcc_install/gcc-build/build/bin.**

```
cd GroupTuner
chmod +x gcc_install.sh
find GroupTuner/Dataset/dataset/cBench_V1.1 -type f -
    name "__run" -exec chmod +x {} \;
./gcc_install.sh
```

- **Create a virtual environment and install necessary packages.**

```
conda create -n grouptuner python=3.9
conda activate grouptuner
pip install -r requirements.txt
export PYTHONPATH=$(dirname "$PWD"):$PYTHONPATH
```

A.6 Experiment workflow

1. Data Preparation: We provide the datasets `cBench` and `PolyBench` under the directory `GroupTuner/Dataset/dataset`.

Due to the complexity and long runtime of SPEC CPU2017 benchmarks, we did not include them in the artifact.

2. Running Experiments: The full experimental execution logic is encapsulated in main_cbench.py and main_poly.py, which respectively perform iterative tuning using GroupTuner and other methods (SA, SRTuner, CFSCA, BOCA, RIO) on cBench and PolyBench. As iterative tuning is time-consuming, to facilitate faster evaluation, we provide a subset of 3 cBench programs (automotive_qsort1, bzip2d, network_dijkstra) and 3 PolyBench programs (cholesky, floyd-warshall, gemm) by default in the experimental scripts. If desired, reviewers can modify the script to evaluate additional benchmarks beyond the provided subset.

To run the experiments, use the following commands:

```
python3 main_cbench.py --gcc-path $PWD/gcc_install/gcc-
    build/build/bin/gcc --round 500
python3 main_poly.py --gcc-path $PWD/gcc_install/gcc-
    build/build/bin/gcc --round 500
```

Here, gcc-path specifies the path to the customized GCC version used for the experiments, and round specifies the number of tuning iterations.

We recommend setting round to 500 for sufficient convergence (as used in our experiments), but you may adjust it according to your available resources. As dynamic iteration is time-consuming, we suggest executing experiments on multiple servers in parallel. However, we strongly recommend that the tuning results of different methods for the same program be collected on the same server to avoid performance variations caused by hardware differences.

Note that the tuning process is continuous and cannot be interrupted. If an experiment is accidentally interrupted, please rerun the experiment for that method from the start.

If you wish to evaluate additional benchmarks, please obtain the full benchmark list from Dataset/dataset/cbench_prog_info.csv and Dataset/dataset/poly_prog_info.csv, and modify the variable test_cases in main_cbench.py and main_poly.py.

3. Result Collection: The experimental results are stored under GroupTuner/Dataset/output/: - cbench/test/: Tuning results for cBench programs. - polybench/test/: Tuning results for PolyBench programs.

4. Results Visualization:

We provide scripts under GroupTuner/show_result to automatically reproduce all figures and tables reported in the Evaluation Section. The scripts automatically collect and summarize results from GroupTuner/Dataset/output.

- **Table 3 (Main Results)**

  ```
  cd show_result
  python3 Table3.py
  ```

  The result will be saved to GroupTuner/show_result/Table3.csv.

- **Figure 3 Results**

  ```
  cd show_result
  python3 Fig3.py
  ```

  The figure will be saved as GroupTuner/show_result/Fig3.png.

- **Figure 4 Results**

  ```
  cd show_result
  python3 Fig4.py
  ```

  Since Figure 4 analyzes the tuning results for each individual program, the outputs are saved separately under GroupTuner/show_result/Fig4_result/, with one figure per program.

- **Figure 5 Results**

  ```
  cd show_result
  python3 Fig5.py
  ```

  Similarly, the outputs are saved under GroupTuner/show_result/Fig5_result/, with one figure per program.

- **Figure 6 Results**

  ```
  cd show_result
  python3 Fig6.py
  ```

  The figure will be saved as GroupTuner/show_result/Fig6.png.

- **Figure 7 Results**

  ```
  cd show_result
  python3 Fig7.py
  ```

  The figure will be saved as GroupTuner/show_result/Fig7.png. The time-consuming analysis for Figure 7 is based on the runtime of the RIO algorithm. Please ensure that the RIO results exist under both Dataset/output/cbench/test/RIO/ and Dataset/output/polybench/test/RIO/ before running Fig7.py.

A.7 Evaluation and expected results

Upon completing the evaluation workflow, the following outputs will be generated:

- Experimental Results: All tuning results are stored under the directory GroupTuner/Dataset/output/. These files record the performance improvements and detailed execution processes obtained by GroupTuner and other methods (SA, SRTuner, CFSCA, BOCA, RIO) across multiple iterations.
- Visualization Results: After collecting the experimental outputs, the corresponding visualization figures and tables are generated under the directory GroupTuner/show_result/. The provided scripts automatically summarize the results from GroupTuner/Dataset/output/ and generate the Tables and Figures.

A.8 Experiment customization

We support the following customization options for users who wish to adapt the experiments to different settings:

- Replacing the Compiler Version: To change the compiler used in the tuning experiments, simply modify the --gcc-path parameter when running main_cbench.py or main_poly.py. Specify the path to the new GCC binary as follows:

  ```
  python3 main_cbench.py --gcc-path /path/to/new/
      gcc --round 500
  python3 main_poly.py --gcc-path /path/to/new/gcc
      --round 500
  ```

  No additional code modifications are required.
- Replacing the Benchmark Dataset: To tune a new dataset:
  1. Implement the data processing logic for the new dataset by modifying GroupTuner/Dataset/process/heads.py.
  2. Update main_cbench.py to replicate the tuning logic for cBench benchmarks with the corresponding logic for the new dataset. You can refer to how each algorithm is currently invoked for cBench to replicate a similar structure for the new dataset.

A.9 Methodology

Submission, reviewing and badging methodology:

- `https://www.acm.org/publications/policies/artifact-review-and-badging-current`
- `https://cTuning.org/ae`