Quantifying Cache Side-Channel Leakage by Refining Set-Based Abstractions (Artifact)

Jacqueline L. Mitchell
University of Southern California, Los Angeles, CA, USA

Chao Wang
University of Southern California, Los Angeles, CA, USA

Abstract

This is the accompanying artifact for the ECOOP 25' paper "Quantifying Cache Side-Channel Leakage by Refining Set-Based Abstractions".

2012 ACM Subject Classification Software and its engineering → Software verification and validation; Theory of computation → Program analysis

Keywords and phrases Abstract interpretation, side-channel, leakage quantification, cache

Digital Object Identifier 10.4230/DARTS.11.2.18

Related Article Jacqueline L. Mitchell and Chao Wang, "Quantifying Cache Side-Channel Leakage by Refining Set-Based Abstractions", in 39th European Conference on Object-Oriented Programming (ECOOP 2025), LIPIcs, Vol. 333, pp. 22:1–22:28, 2025.
`https://doi.org/10.4230/LIPIcs.ECOOP.2025.22`

Related Conference 39th European Conference on Object-Oriented Programming (ECOOP 2025), June 30–July 2, 2025, Bergen, Norway

Evaluation Policy The artifact has been evaluated as described in the ECOOP 2025 Call for Artifacts and the ACM Artifact Review and Badging Policy.

1 Scope

The scope of the artifact is intended to cover reproducing the results from Table 1, Table 2, and the ablation studies from Figure 9 of the associated paper. Table 1 is intended to empirically evaluate how the technique improves upon the state of the art tool on one cache configuration. Table 2 demonstrates the synergistic effect of the two techniques on a representative program. Figure 9 compares compares each technique to the baseline tool, on a wide range of cache configurations.

2 Content

The artifact package includes a Docker image containing the following:

- The source code used to implement the techniques described in the paper (located in `/app/quantitative_cache_analysis`)
- The list of benchmark programs (and their corresponding locations within the Docker image) used in the experiments section of the paper can be found at `quantitative_cache_analysis/run_experiments/all_programs.txt`.
- Scripts used to:
  1. Reproduce (all or a subset of) the results in Table 1.
  2. Reproduce the results in Table 2.
  3. Reproduce (all or a subset of) the results in the scatter plot of Figure 9.

  (Located in the directory: `quantitative_cache_analysis/run_experiments`)

3 Getting the artifact

The artifact endorsed by the Artifact Evaluation Committee is available free of charge on the Dagstuhl Research Online Publication Server (DROPS). In addition, the artifact is also available at: `https://doi.org/10.6084/m9.figshare.28890416`.

A quick-start guide for using the artifact follows:

1. Install Docker (if not already installed).
2. Load the Docker image: `docker load -i quantitative_cache_analysis_ecoop25.tar`
3. Run an artifact container: `docker run -it ecoop25-cache-analysis:v1`
4. Navigate to the main directory: `cd quantitative_cache_analysis`
5. Navigate to the directory where experiments are run: `cd run_experiments`
6. Run the smoke test to make sure everything's working.
   a. Clean-up the working directory first: `./clean_up_experiment_state.sh 1 1 1`
   b. Run the smoke test: `./smoke_test.sh` (This should take a couple of minutes).
   c. Verify that a table is shown in the terminal output and that the working directory is cleaned after the table is shown (shown through log output).

4 Tested platforms

This artifact has been tested on a machine with an Intel Xeon W-2245 CPU with 128 GB RAM, with 500GB of disk space, running the Ubuntu 20.04 operating system (the full set of experiments shown in the paper were run on a machine with this specification). The artifact also has been tested for functionality on a machine running Ubuntu 24.04, with an Intel(R) Core(TM) Ultra 9 185H CPU with 64 GB RAM and 1TB of disk space.

All of the experiments should be runnable on a reasonably powerful laptop. However, some of the experiments may take longer (e.g. 2 days). For the more intensive experiments, we provide a way to compute a subset of the results.

5 License

The artifact is available under the MIT License. (See `LICENSE.txt` at `https://doi.org/10.6084/m9.figshare.28890416`)

6 MD5 sum of the artifact

e4c4021f0eb262f8a18d59d56eeba437

7 Size of the artifact

2.14 GiB

A Cleaning Up the Working Directory

To clean up the by-products of generating the experimental results, the following steps can be taken:

1. Run `./clean_up_experiment_state.sh [1 | 0] [1 | 0] [1 | 0]`
   a. The first command line argument indicates whether or not to clean up the experimental results generated by `./generate_table_1_results.sh`.

   b. The second command line argument indicates whether or not to clean up the experimental results generated by `./generate_table_2_results.sh`.
   c. The third command line argument indicates whether or not to clean up the experimental results generated by `./generate_figure_9_results.sh`.

B Reproducing Table 1

The data in Table 1 can be obtained using the following steps. The commands that should be run are highlighted in blue.

1. Ensure that the working directory is clean (w.r.t. results for Table 1): `./clean_up_experiment_state.sh 1 0 0`
2. Run `./generate_table_1_results.sh [true | false] [.txt]`
   a. The first argument indicates whether or not to generate results for the baseline tool.
   b. The second argument is a `.txt` file containing a list of programs (benchmarks) for the tool to run. The full set of programs, stored in a `.txt` file, can be found in `all_programs.txt`. Note that if the full set of programs are run along with the baseline tool, this may take a few hours to complete. The reviewers also have access to a subset of the programs featured in Table 1 (`subset_programs.txt`). With the provided set, the experiment (including running the baseline) takes less than 10 minutes to run. The reviewers are welcome to remove or add programs from `all_programs.txt` to this subset as they see fit.
   c. NOTE: The timeout for each individual run (one program + one cache configuration) (called from within `generate_table_1_results.sh`) is set to one hour. The exception is for runs that involve the `hc-128` program, which has a timeout of 1.5 hours.
   d. The script works in two phases: (1) The script generates the raw results. (2) The script parses the raw results and generates the output. The script is set up such that if the raw results already exist within the directory, the raw results are not re-generated, and the existing results are simply parsed and displayed.
   e. Here is the command to run both the baseline and the new method on the subset of programs: `./generate_table_1_results.sh true subset_programs.txt`
   NOTE: please ensure that both command line arguments are provided to the script, otherwise it will result in an error.
   NOTE: please also ensure that when programs are added to `subset_programs.txt`, that the entire line associated with the program is copied over entirely from `all_programs.txt`, or it may result in an error.
   f. **Once the command is run, the table should be printed in the terminal. A `.txt` version of the file will be stored in `table_1_final_results.txt`.**

C Reproducing Table 2

The data in Table 2 can be obtained using the following steps. The commands that should be run are highlighted in blue.

1. Ensure that the working directory is clean (w.r.t. results for Table 2): `./clean_up_experiment_state.sh 0 1 0`
2. Run `./generate_table_2_results.sh`
   a. This should take about ten minutes to run.

   b. NOTE: The script works in two phases: (1) The script generates the raw results. (2) The script parses the raw results and generates the output. The script is set up such that if the raw results already exist within the directory, the raw results are not re-generated, and the existing results are simply parsed and displayed.
3. **Once the command has completed running, the data from Table 2 should be printed in the terminal. A `.txt` version of the file should be saved to `table_2_final_results.txt`.**
4. NOTE: The following errors that may show up in the log as the program is running are normal:
   a. `Stack_overflow` – This error occurs when running the given cache configuration is too memory-intensive.
   b. `Detected Top in set error, retrying with larger unroll value...` – Some programs may require loop-unrolling. If this error occurs, the loop is unrolled and the benchmark is ran again.
   c. `Timeout Errors` – A given run (program + cache configuration) may time out.

D Reproducing Figure 9

The scatter plot in Figure 9 can be obtained using the following steps. The commands that should be run are highlighted in blue.

1. Ensure that the working directory is clean (w.r.t. results for Figure 9):
   `./clean_up_experiment_state.sh 0 0 1`
2. Run `./generate_figure_9_results.sh [.txt]`
   a. The argument is a `.txt` file containing which programs (benchmarks) to use in the generation of the plot.
   b. To completely recreate the plot in Figure 9, the command should be run with the argument `all_programs.txt`. We note that this may take a very long time (1-2 days) to run. Thus, the reviewers may choose to run on a restricted subset of programs, provided in `subset_programs.txt`. With the provided set, the experiment takes around 40 minutes to run to completion. The reviewers are welcome to remove or add programs from `all_programs.txt` to this subset as they see fit.
   c. To run the subset of programs: `./generate_figure_9_results.sh subset_programs.txt`
   NOTE: please ensure that the command line argument is provided to the script, otherwise it will result in an error will.
   NOTE: please also ensure that when programs are added to `subset_programs.txt`, that the entire line associated with the program is copied over entirely from `all_programs.txt`, or it may result in an error.
3. **Once the command has completed running, (a subset of) the data from Figure 9 should be printed in the terminal. A .png file of the scatterplot should be saved to `figure_9_plot.png`.**
4. NOTE: The following errors that may show up in the log as the program is running are normal:
   a. `Stack_overflow` – This error occurs when running the given cache configuration is too memory-intensive.
   b. `Detected Top in set error, retrying with larger unroll value...` – Some programs may require loop-unrolling. If this error occurs, the loop is unrolled and the benchmark is ran again.
   c. `Timeout Errors` – A given run (program + cache configuration) may time out.