USENIX'23 Artefact Appendix: CAPSTONE: A Capability-based Foundation for Trustless Secure Memory Access

Jason Zhijingcheng Yu
National University of Singapore

Conrad Watt
University of Cambridge

Aditya Badole
National University of Singapore

Trevor E. Carlson
National University of Singapore

Prateek Saxena
National University of Singapore

A Artefact Appendix

A.1 Abstract

This artefact includes the following components:

- Functional prototypes of CAPSTONE. More specifically, those include the emulator CAPSTONEEmu, the compiler CAPSTONECC, and the library CAPSTONELib, along with sample source codes for the case studies discussed in the paper that are runnable with the aforementioned tools. This part resides under the `functional` subfolder.
- The GEM5 model used for evaluating CAPSTONE. This includes the source code and the scripts for building both the model and the benchmarks as well as for running the experiments presented in the paper. This part resides under the `gem5` subfolder.

All the artefact components have been made publicly available in the source format. To improve portability, reduce the impact on the artefact user's own system, and ease the process of using the artefact itself, we provide the option of building and running the artefact inside Docker containers.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

Building and running this artefact are not expected to cause security or privacy risks to the artefact user. Nor is it expected to raise ethical concerns.

A.2.2 How to access

The artefact is available on Github at https://github.com/jasonyu1996/capstone (revision hash: 9b5319c). Note that this Github repository includes submodules. To download all the included components, make sure that you supply `--recurse-submodules` when you clone it, or run

```
git submodule update --init --recursive
```

afterwards.

A.2.3 Hardware dependencies

None.

A.2.4 Software dependencies

The platform to use this artefact on is expected to have Bash installed and support running x86-64 Docker containers (Docker version 20 or later recommended).

A.2.5 Benchmarks

For copyright reasons, we have not included SPEC CPU 2017, the benchmark suite used for the evaluation experiments with the GEM5 model. The user needs to supply the benchmark suite by themselves if they wish to run those experiments.

A.3 Set-up

We have included detailed instructions in the `README.md` files in the Github repository. Below we only reproduce the brief steps.

A.3.1 Installation

Functional prototypes Change the working directory to `functional`. Build the Docker image with

```
./build
```

GEM5 model Change the working directory to `gem5`. Build the GEM5 model for Capstone and the baseline model with

```
./run-docker build
```

Note that the above command will pull `corank/gem5-dev` if the Docker image does not exist locally. You can pull it manually with

```
docker pull corank/gem5-dev
```

or alternatively, build it on your own machine

```
cd docker-build
docker build . -t corank/gem5-dev
```

To build SPEC CPU 2017, place it under `./spec` and apply a patch before running the build script

```
(cd spec && patch -p1 \
< ../tests/capstone/speckle/spec17.patch)
./run-docker build-spec
```

A.3.2 Basic test

Functional prototype Test with

```
./run compiler/samples/dummy.c
```

You should be able to see the output which starts with

```
18: GPR 1 = Value 0
19: halted
```

followed by runtime statistics.

GEM5 model Run

```
./run-docker run-hello
```

The output should include

```
Hello gem5!
```

A.4 Evaluation Workflow

The evaluation workflow applies to the GEM5 model only.

A.4.1 Major Claims

(C1): In comparison to the baseline RISC-V model, the GEM5 model for CAPSTONE exhibits overhead that ranges from 0 to 50% across SPEC CPU 2017 workloads (as shown in Figure 3 in the paper).

A.4.2 Experiments

(E1): estimated 30 compute-hours (when running workloads in parallel):
How to: Please follow the following steps to run this experiment.
Preparation: Build both the GEM5 model and the benchmark suite SPEC CPU 2017 following the steps described in Section A.3.1.
Execution: Run SPEC CPU 2017 with the GEM5 model for CAPSTONE first

```
./run-docker run-capstone --multiproc
```

followed by the baseline RISC-V model

```
./run-docker run-baseline --multiproc
```

Note that the `--multiproc` flag can be omitted, but that will result in the experiments being run on a single CPU core, which would be slow and hence not recommended.
Results: The logs are available in `./outputs`. To parse the logs and produce the data shown in Figure 3 in the paper,

```
./run-docker collect-results
```

which prints to the standard output the parsed results in the LaTeX table format.

A.5 Notes on Reusability

The behaviours of the compiler CAPSTONECC can be adjusted through command line flags. Please read the source code `compiler/src/main.rs` or `README.md` for details.

For the GEM5-based evaluation, it is possible to change the number of fast-forwarded instructions, and the number of instructions to simulate after fast-forwarding. This is achieved by adjusting the variables `GEM5_SKIP` and `GEM5_LIM` in scripts `docker-scripts/run-capstone` and `docker-scripts/run-baseline`. Similarly, the size of the node cache can be set through the variable `GEM5_NCACHE`. To print more data, set `GEM5_FLAGS` to `--debug-flags=...` with the debug flags defined in GEM5.

A.6 Version

Based on the LaTeX template for Artefact Evaluation V20220926. Submission, reviewing and badging methodology followed for the evaluation of this artefact can be found at https://secartefacts.github.io/usenixsec2023/.