gem5art: Artifact, Reproducibility and Testing Framework for gem5

Ayaz Akram, Mahyar Samani, Hoa Nguyen, Krithiga Murugavel, Trivikram Reddy, Marjan Fariborz, Pouya Fotouhi, and Jason Lowe-Power

University of California, Davis

Workshop on Modeling & Simulation of Systems and Applications (ModSim2020), 2020

gem5art: Artifact, Reproducibility and Testing Framework for gem5

Ayaz Akram, Mahyar Samani, Hoa Nguyen, Krithiga Murugavel, Trivikram Reddy,
Marjan Fariborz, Pouya Fotouhi, Jason Lowe-Power

The gem5 simulator[1] is one of the most popular computer architecture research simulation frameworks with its multitude of hardware models and rich support for full system simulation. However, it has a steep learning curve and using full system simulation requires an arduous set up. A typical simulation requires preparing benchmarks, a kernel, a disk image with both an operating system and the prior mentioned benchmarks, the simulator binary, and the simulator configuration.

The goal of gem5art is to simplify this procedure and give experimenters an easy and standard process to document every part of their experiments. The gem5art framework provides a systematic method for describing an experimental protocol when using the gem5 simulator. To use the open source gem5art framework, gem5 users can download the Python libraries from the Python Package Index (PyPI)[2] and create a new gem5art script which describes their experimental setup.

The gem5art framework[3] contains libraries for *artifacts*, *reproducibility*, and *testing*. When running an experiment, there are inputs, steps to run the experiment, and outputs. gem5art tracks each of these through artifacts. An artifact is some object, usually a file, which is used as part of the experiment. gem5art relies on multiple attributes (e.g. hash, UUID, name, type) of an artifact to maintain its uniqueness. Additionally, gem5art uses a database to store these artifacts which enables reproducibility and sharing artifacts with others doing similar experiments (e.g., a disk image with a shared workload). Another goal of gem5art is to provide an extendable base setup to run a variety of experiments and enable cross-experiment analysis with ease.

Our group has been using gem5art for about 6 months and have a total of over 3500 experiments run with gem5art. We have also used gem5art to test the gem5-20 release candidate and found multiple bugs before the final release. We have created the artifacts necessary to run many popular benchmark suites (e.g., SPEC 2006, SPEC 2017, Parsec, NAS Parallel Benchmarks, the GAP Benchmarks Suite, and microbenchmarks for testing components), popular operating systems (e.g., Ubuntu), and multiple Linux kernel versions. These artifacts will be freely available on the gem5 website as part of the "gem5-resources" repository (when their licenses allow). The gem5art framework includes a description of how each artifact is created, and we also provide detailed documentation on gem5art, a set of tutorials for each of the provided workloads[4], and a git template repository to bootstrap gem5art-based experiments.

Evaluation: We present few observations (based on our experience) which highlight how gem5art has been able to achieve its goals referred previously. Carrying out experiments with gem5art relies on Python launch scripts which encompass documenting artifacts used for an experiment to launch an actual gem5 job (all in a standardized workflow). The average length of these scripts (for benchmark suites discussed above) is 153 lines of code, most of which correspond to filling out the user-defined attributes of the created artifacts. These launch scripts also help to reduce the time required to extend an experiment to use a different configuration. For example, extending the Linux kernel boot experiments to a newer kernel version or new simulator configurations only takes a few minutes. Similarly, we found the time to build a disk image (with benchmarks and runtime) to be reduced significantly and the process to be less error-prone, as it is scripted in gem5art framework using packer (a disk image creating tool).

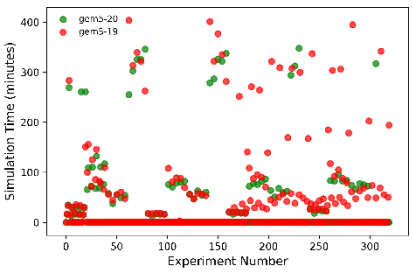

Figure 1: Example meta-study. Simulator performance for two gem5 versions.

Due to gem5art's ability to store experimental results in a database, it allowed us to easily do cross experimental studies. Figure 1 shows the simulation time of two different versions of gem5 (gem5-19 and gem5-20) for different simulator configurations while booting different Linux kernels (each position on the x-axis is a different configuration). This cross-experiment study enables us to visualize not only if the working status of the simulator across two of its versions has changed, but also if there are any changes in the performance of the simulator. For example, as shown in the figure, some high time-consuming red dots do not have green counter parts, which corresponds to some failing simulations with a detailed CPU model in gem5-20 (which were working with gem5-19). Secondly, the green dots are generally slightly lower than the pairing red dots indicating that the performance of gem5-20 is slightly better than gem5-19. Though it was not impossible to do such studies before, with gem5art such studies are much easier.

[1] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, et al., "The gem5 Simulator," ACM SIGARCH Computer Architecture News, vol. 39, pp. 1– 7, May 2011.
[2] https://pypi.org/
[3] https://github.com/darchr/gem5art/
[4] https://gem5art.readthedocs.io/en/latest/