GKLEE: Practical Concolic Verification and Test Generation for GPUs

Peng Li, Guodong Li, Ganesh Gopalakrishnan
School of Computing, University of Utah, Fujitsu Laboratories of America
{peterlee,ganesh}@cs.utah.edu, gli@cs.utah.edu

GKLEE-generated tests can be run on the actual hardware, tracking GKLEE’s virtual machine incorporates the CUDA hierarchical memory

GKLEE employs powerful heuristics for reducing the number of tests

GKLEE is the first concolic (concrete plus symbolic) verifier and test generator tailored for GPU programs, and provides thread execution features of several Nvidia GPU families (compute capabilities).

GKLEE analyzes GPU programs with respect to important correctness and performance issues.

GKLEE can systematically generate concrete tests that provide high coverage.

GKLEE employs powerful heuristics for reducing the number of tests while retaining high coverage.

GKLEE-generated tests can be run on the actual hardware, tracking the consistency between GKLEE’s performance predictions and results obtained from commercial performance tools.

GKLEE’s virtual machine incorporates the CUDA hierarchical memory model within its concolic execution framework, while (i) accurately modeling the SIMD concurrency of GPU programs, (ii) avoiding interleaving enumeration, and (iii) scaling to large code sizes.

Given a C++ program, the GKLEE virtual machine executes the following steps, in order, for each control-flow path pursued during execution:

- Create the GPU memory objects representing the code memory regions representing GPU memory dynamically.
- Execute GPU kernel threads via the concolic execution framework, while (i) accurately modeling the SIMD concurrency of GPU programs, (ii) avoiding interleaving enumeration, and (iii) scaling to large code sizes.
- Fork new states upon non-determinism due to symbolic memory races (if any), perform test-case selection, and write out a concrete test file.

In a state, at the end of the barrier interval or other synchronization points, perform checks for deadlocks, data memory accesses, volatile qualifier missing, and performance issues.

Future work:
Besides the parameterized concolic verification, our future work will exploit the symbolic equivalence checking to consolidate the functional correctness verification and explore an automatic bug-preserving kernel downsizing method, capitalizing on symmetries, tighter integration of CUDA library calls, and analysis of kernels utilizing CUDA atomic and stream calls, in addition to solving all the examples in popular CUDA textbooks.

Reference