Accurate Profiling in the Presence of Dynamic Compilation

Many programming languages are implemented on top of a managed runtime system, such as the Java Virtual Machine (JVM) or the .NET CLR, featuring an optimizing dynamic (just-in-time) compiler. Programs written in those languages are first interpreted (or compiled by a baseline compiler), whereas frequently executed methods are later compiled by the optimizing dynamic compiler. Common feedback-directed optimizations performed by state-of-the-art dynamic compilers, such as the optimizing compiler in the Jikes RVM or Graal, include method inlining and stack allocation of objects based on (partial) escape analysis, amongst others. Such optimizations result in compiled machine code that does not perform certain operations present at the bytecode level. In the case of inlining, method invocations are removed. In the case of stack allocation, heap allocations are removed and pressure on the garbage collector is reduced.

Many profiling tools are implemented using bytecode instrumentation techniques, inserting profiling code into programs at the bytecode level. However, because dynamic compilation is transparent to the instrumented program, a profiler based on bytecode instrumentation techniques is not aware of the optimizations performed by the dynamic compiler. Prevailing profilers based on bytecode instrumentation suffer from two serious limitations: (1) over-profiling of code that is optimized (and in the extreme case completely removed) by the dynamic compiler, and (2) perturbation of the compiler optimizations due to the inserted instrumentation code.

We present a novel technique to make profilers implemented with bytecode instrumentation techniques aware of the optimization decisions of the dynamic compiler, and to make the dynamic compiler aware of inserted profiling code. Our technique enables profilers which collect dynamic metrics that (1) correspond to an execution of the base program without profiling (w.r.t. the applied compiler optimizations), and (2) properly reflect the impact of dynamic compiler optimizations.

We implement our approach in a state-of-the-art Java virtual machine and demonstrate its significance with concrete profilers. We quantify the impact of escape analysis on allocation profiling, object lifetime analysis, and the impact of method inlining on callsite profiling. We illustrate how our approach enables new kinds of profilers, such as a profiler for non-inlined callsites, and a testing framework for locating performance bugs in dynamic compiler implementations.

Author: Walter Binder (walter.binder@usi.ch)