An Empirical Study on Deoptimization in the Graal Compiler

Managed language platforms such as the Java Virtual Machine or the Common Language Runtime rely on a dynamic compiler to achieve high performance. Besides making optimization decisions based on the actual program execution and the underlying hardware platform, a dynamic compiler is also in an ideal position to perform speculative optimizations. However, these tend to increase the compilation costs, because unsuccessful speculations trigger deoptimization and recompilation of the affected parts of the program, wasting previous work. Even though speculative optimizations are widely used, the costs of these optimizations in terms of extra compilation work has not been previously studied. In this paper, we analyze the behavior of the Graal dynamic compiler integrated in Oracle's HotSpot Virtual Machine. We focus on situations which cause program execution to switch from machine code to the interpreter, and compare application performance using three different deoptimization strategies which influence the amount of extra compilation work done by Graal. Using an adaptive deoptimization strategy, we managed to improve the average start-up performance of benchmarks from the DaCapo, ScalaBench, and Octane benchmark suites, mostly by avoiding wasted compilation work. On a single-core system, we observed an average speed-up of 6.4% for the DaCapo and ScalaBench workloads, and a speed-up of 5.1% for the Octane workloads; the improvement decreases with an increasing number of available CPU cores. We also find that the choice of a deoptimization strategy has negligible impact on steady-state performance. This indicates that the cost of speculation matters mainly during start-up, where it can disturb the delicate balance between executing the program and the compiler, but is quickly amortized in steady state.

Contact: walter.binder@usi.ch