Title: Interpreting the Interpreter Just-in-time
Name: Jonathan Lambert
Supervisors: Prof. Rosemary Monahan, Dr Kevin Casey

Abstract

The benefits associated with interpretive execution are well recognised; for example, interpreters are more adaptable, simpler, more portable than compilers, they have a compact code footprint, reduced hardware complexity, reduced development costs, are more manageable regarding debugging and profiling, and expensive tasks such as register allocation and call-stack management are unnecessary. Also, they play an integral part in more recent execution models, such as the tiered execution methodology introduced within the Java runtime environment's (JRE) virtual machine, which relies upon code interpretation within its first tier of execution, providing quick start-up performance and subsequently detailed dynamic profiling information of the application to guide just-in-time compilation of the bytecode within subsequent execution tiers. Nevertheless, the advantages of interpretive execution may become obscured when considering the execution time metric, as just-in-time execution provides significantly better execution time efficiency compared to interpretation. With that said, important contemporary questions regarding the performance advantages of interpretive execution over tiered execution remain unanswered, particularly regarding the average and peak power consumption metrics. This talk addresses those questions through a multifaceted analysis of the power consumption behaviour of the JRE and its associated Java Virtual Machine (JVM). In particular, we model power consumption differences between JRE versions, the JRE build toolchain compiler versions used in the building of the JRE, the effects of changing garbage collector, and the effects that workload type has on power consumption. In addition, we model the effects that microarchitectural performance indicators have on power consumption. The analysis is relative to the performance measures captured from 196 OpenJDK JRE builds and their associated JVMs. Our investigations have identified many important factors that influence power consumption. In general, our findings have shown that it should not be presumed that modern tiered execution will outperform interpretive execution, in fact, there exist a class of workloads for which interpretive execution provides better average power consumption efficiency, and in general better peak power consumption efficiency. Our findings have also shown instances of significant regression in JRE performance, with that said, excluding those outliers, no real change in performance is notable across versions. The "Update JVM, run the same application, realise improved performance," argument does not necessarily hold. Our findings have shown that real consideration needs to be given to the JRE source version, build type, execution mode, garbage collection strategy, and more importantly the type of workload that the JRE JVM will be executing. In that regard, our findings would provide evidence for a change in execution model, with the parking of JREs and their associated JVMs for use in a just-in-time manner. By examining the JRE version, GCC version, garbage collector, workload, and a range of microarchitectural metrics, we assessed the ability of those factors to predict power consumption. Our findings revealed that approximately 75% of the fluctuation in power consumption is explained by the variances in these predictors. Moreover, by employing a generalized additive model that accommodates non-linear patterns, our model accounts for approximately 92% of the variation in power consumption.

Keywords: Java Runtime Environment; Java Virtual Machine; JVM Template Interpreter, JRE Performance, GNU GCC Compiler Collection, GNU GCC Effect on JRE Performance, Garbage Collector Effect on JRE Performance, OpenJDK, Compiler Optimisations, Renaissance Benchmark Suite