An Analysis of Specialized Java Bytecodes

Ben Stephenson

The Java programming language offers a variety of useful language features including an extensive library, automatic memory management, object oriented language features and platform independence. While these features are designed to ease application development and improve programmer productivity, some of these features have performance costs associated with them. In particular, platform independence is normally accomplished using a virtual machine which can impose large overheads, hurting the runtime performance of Java applications. As a result, gaining insight into the nature of Java applications is valuable as it may lead to the development of new optimization techniques that help minimize the amount of inefficiency incurred.

When a Java application is compiled, it is converted from human readable Java source code to a sequence of Java bytecodes. The Java Virtual Machine specification defines the functionality of 201 distinct bytecodes. When the application is executed, the Java Virtual Machine reads the bytecodes stored in a Java class file and executes the program by performing the actions required by each bytecode encountered.

Some of the bytecodes specified by the Java Virtual Machine Specification provide unique functionality that is not easily duplicated by other bytecodes. Other bytecodes perform tasks that are easily mimicked by either one other bytecode or a short sequence of bytecodes. Those bytecodes that provide unique functionality will be referred to as general bytecodes while the remaining bytecodes will be referred to as specialized bytecodes.

Over 70 specialized bytecodes have been identified. Previous research has categorized the despecializations based on the nature of the operations needed to replace the specialized bytecode. This work also included performance testing where each category was despecialized, generally resulting in a minor performance loss for the application.

This presentation extends previous work by considering only a subset of the specialized bytecodes. Profiling was used to identify those specialized bytecodes that occurred with the least frequency. Such bytecodes were removed from the Java class files in both the application and Java library. Performance testing showed only minor differences in application run time.

Previous work is further extended by identifying new specialized bytecodes. Profiling has shown that some general bytecodes are executed frequently with specific operands. In fact, several general bytecodes are executed with a given operand more frequently than some specialized bytecodes. This result indicates that better choices could have been made when the designers of the Java Virtual Machine Specification decided what specialized bytecodes would be included. Performance results are presented showing the performance impact of using different specialized bytecodes. Further results are presented when both infrequently occurring specialized bytecodes are removed and when new specialized bytecodes are introduced.