An Introduction to Multicodes
Ben Stephenson

A number of studies have been performed which characterize the behaviour of Java applications. The simplest of these studies only consider the static nature of the application, reporting statistics such as the total number of classes, methods and bytecodes. These measures are ineffective because they consider parts of the program that are never executed and do not give additional weight to portions of the program that execute many times.

Dynamic analysis techniques were developed which characterize the behaviour of the application based on values collected at runtime. These measures are more meaningful because they only consider portions of the program that are actually executed, ignoring unused or unreachable code. Some examples of dynamic statistics include the number of classes loaded, the number of distinct methods executed and the total number of method invocations. Dynamic measures can also be determined at the Java bytecode level. For example, a bytecode level dynamic analysis can report the number of times a particular operation is performed by the application, the total number of times a bytecode that accesses a local variable is executed and the number of reads from static variables.

Multicode analysis extends bytecode level dynamic characterization by considering sequences of bytecodes executed by an application rather than considering each bytecode in isolation. This has shown that some sequences of bytecodes are extremely common. For example, when sequences of two bytecodes are considered, the pair `aload_0 getfield` was most common. This is not surprising because this sequence is used to load the value of a field onto the operand stack so that additional operations can be performed on it.

When larger multicodes are considered, the amount of commonality across applications decreases. However, when a single application is considered, multicode analysis reveals that there are many longer bytecode sequences that occur frequently. Some commonly occurring sequences are presented in addition to the algorithms used to determine the best bytecode sequences based on their frequency, length, and optimization potential. It is interesting to note that several longer sequences occur in the top 10 bytecode sequences for each application.

Implementing multicodes has revealed that they can be used to improve the performance of Java applications. When the most common 2 bytecode sequence, `aload_0 getfield`, was replaced with a multicode, a performance gain of between 1.5 and 4.0 percent was observed depending upon the benchmark executed. When five commonly occurring 2 bytecode sequences were replaced with multicodes, performance gains of more than 4.0 percent were achieved for all benchmarks except one that showed a performance gain in excess of 7.0 percent.

Performance measurements for the most common multicodes for a specific application have also been determined. Some applications have shown performance gains of almost 10 percent when ten bytecode sequences were replaced by multicodes while other applications have shown no significant performance benefit. The reason for this anomalous behaviour is not yet known but a number of theories are presented.

Future research directions will be discussed including techniques for optimizing multicodes to achieve further performance gains. A number of possible implementation strategies will also be outlined.