An Evaluation of Specialized Java Bytecodes

Ben Stephenson
Department of Computer Science
University of Western Ontario
London, Ontario, Canada
ben@csd.uwo.ca

Wade Holst
Department of Computer Science
University of Western Ontario
London, Ontario, Canada
wade@csd.uwo.ca

Abstract

Specialized Java bytecodes provide functionality that is easily replicated using other Java bytecodes. This study uses profiling to explore how the set of specialized bytecodes currently implemented by the Java Virtual Machine is utilized by comparing it to the other specialized bytecodes which could have been implemented.

Categories and Subject Descriptors D.3.4 [*Programming Languages*]: Optimization

General Terms Performance, Languages

Keywords virtual machine, bytecode, optimization, Java

1. Introduction

The Java Virtual Machine executes platform independent, binary, Java class files that represent a Java application. Within these class files, the functionality of each method is described using an instruction set that consists of 201 distinct operations, known as Java bytecodes. Each bytecode consists of an opcode and zero or more operands.

Previous studies [2] have classified these bytecodes as *specialized* and *general purpose*. A *specialized* bytecode provides functionality that is easily replicated using one other bytecode (or a short sequence of bytecodes) while a *general purpose* bytecode provides unique functionality that cannot be replicated easily. Examples of specialized bytecodes include `fload_3`, `astore_1` and `iconst_1`. These tasks can be performed equivalently using the general purpose opcode/operand pairings `fload 0x03`, `astore 0x01` and `bipush 0x01` respectively.

This study examines the frequency with which general purpose bytecodes are executed with a specific operand value. This frequency is compared with the execution frequency for similar specialized bytecodes. The results of this comparison are illuminating, and suggest changes that could be made to future version of the Java Virtual Machine Specification as well as describing issues that should be considered by researchers developing new intermediate representations.

Copyright is held by the author/owner(s).
OOPSLA'06 October 22–26, 2006, Portland, Oregon, USA.
ACM 1-59593-491-X/06/0010.

2. Benchmarks

This study has considered a total of 23 benchmarks. They include 6 benchmarks from the SPEC JVM98 benchmark suite[1], 12 benchmarks from the Java Grande Forum benchmark suite[2], the SciMark 2.0 benchmark[3], 4 benchmarks from the Ashes Suite Collection[4] and the Linpack numeric benchmark[5]. Each benchmark was executed on a copy of version 1.0.7 of the Kaffe virtual machine that was modified to record each opcode and operand executed.

3. Profile Results

Results for three major categories of bytecodes are presented in this study: local variable load bytecodes, local variable store bytecodes and constant loading bytecodes. The results presented show the frequency with which each specialized byte bytecode in the category is executed and how frequently each general purpose bytecode is executed with a specific operand value.

3.1 Local Variable Loads

The Java Virtual Machine Specification [1] includes 25 bytecodes that load the value of a local variable onto the operand stack. Of these, 20 are specialized bytecodes while 5 are general purpose. Figure 1 shows the distribution of the 20 most executed local variable load bytecodes. Gray bars denote specialized bytecodes while white bars denote general purpose bytecodes executed with a specific operand value.

Examining the graph reveals that general purpose bytecodes account for 12 of the 20 most frequently executed local variable load bytecodes. This result is surprising because it indicates that many general purpose bytecodes are being used with a specific operand with greater frequency than the specialized bytecodes defined by the Java Virtual Machine Specification.

The bytecodes used to load a local variable with least frequency were also considered. It was found that 4 of the 40 least frequently executed local variable load bytecodes were specialized. This is also a surprising result because it indicates that of all of the choices that could have been made for specialized local variable load bytecodes, four of the worst were selected. Such instructions could have been used for the efficient implementation of language features, or removed from the virtual machine to reduce its complexity.

[1] http://www.spec.org/jvm98/
[2] http://www.epcc.ed.ac.uk/javagrande/javag.html
[3] http://math.nist.gov/scimark2/index.html
[4] http://www.sable.mcgill.ca/ashes/
[5] http://www.netlib.org/benchmark/linpackjava/

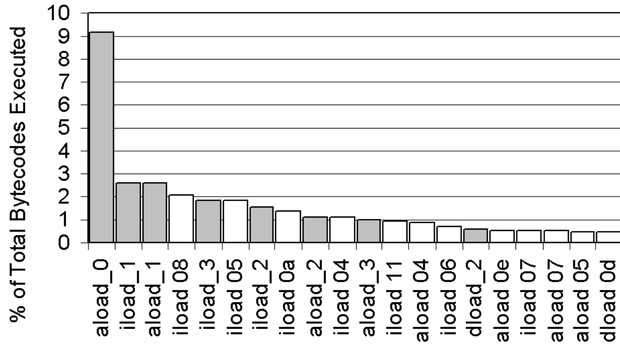

Figure 1. Most Frequently Executed Load Bytecodes

Two design decisions are responsible for the relatively poor use of specialized local variable load bytecodes.

1. The virtual machine instruction set is symmetric with respect to the types handled by specialized bytecodes
2. Method parameters reside in the first local variable slots

Examining the 40 most frequently executed load bytecodes reveals that half of those bytecodes operate on integers. In contrast, only one fifth of the specialized bytecodes defined by the Java Virtual Machine Specification are dedicated to accessing integer local variables. Removing the symmetry from the instruction set by allocating more specialized bytecodes to those types that are used with greatest frequency would improve specialized instruction utilization.

The fact that method parameters reside in the first local variables also reduces the execution frequency of specialized local variable load bytecodes. Because the receiver object is passed as the first parameter to an instance method, a local variable load of non-object reference type can never occur on slot 0 in an instance method. Furthermore, the compiler often does not have the freedom to allocate frequently loaded values to those slots for which specialized bytecodes exist because they are already occupied by method parameters. Solving this problem is complex because any solution should allow the specialized bytecodes to be utilized effectively regardless of the number of parameters passed to the method. As a result, the best option might be to handle local variables and method parameters as separate entities.

3.2 Local Variable Stores

Similar execution patterns were observed for local variable store bytecodes. In particular, examining the 20 most frequently executed store bytecodes showed that 5 of these bytecodes were specialized while the remainder were general purpose. Five specialized store bytecodes were also found among the 40 least frequently executed store bytecodes, including four which were never executed by any of the 23 benchmarks tested.

The limited utilization of specialized store bytecodes occurs for the same reasons that specialized load bytecodes are not fully utilized. In addition, the ability to make effective use of specialized store bytecodes is further complicated by the parameter passing semantics imposed by the Java Virtual Machine. All method parameters are passed by value. As a result, any value stored into a local variable that holds a method parameter will not be visible in the calling scope. Consequently, stores into a local variable that store a method parameter are extremely rare.

3.3 Constant Loads

Unlike the specialized local variable load and store bytecodes, the set of specialized constant loading bytecodes is not symmetric. Of the 14 specialized constant loading bytecodes, 7 are devoted to loading integers in the range -1 to 5, with the remaining 7 devoted to loading constants of type `float`, `long` and `double`. This resulted in better utilization of specialized constant loading bytecodes than the specialized load and store bytecodes considered previously. In particular, 9 of the 20 most frequently executed constant loading bytecodes are specialized,

Examining the 40 least frequently execute constant loading bytecodes revealed that none of the specialized constant loading bytecodes were present in this set.

4. Future Work

Previous work on despecialization has examined the performance impact of removing specialized bytecodes and replacing them with their equivalent general purpose forms. The research presented here reveals that at least part of the reason that small performance changes were observed previously was because many specialized bytecodes are executed relatively infrequently. In the future, we intend to explore what performance gains can be achieved by implementing a new set of specialized bytecodes based on the execution frequencies collected during this study. We speculate that an improvement in performance will be achieved, particularly in smaller Java environments which may not make use of a just-in-time compiler.

5. Conclusion

This study profiled 23 Java benchmarks. The frequency with which specialized bytecodes were executed was compared with the frequency with which general purpose bytecodes were executed with a specific operand value. It was found that specialized local variable load and store bytecodes were not utilized to their full potential, with no more than 8 of 20 specialized bytecodes (40%) appearing among the 20 most frequently executed bytecodes in each category. In contrast, 9 of the 14 specialized constant loading bytecodes (64%) were found among the 20 most frequently executed constant loading bytecodes. We attribute at least part of this difference to the lack of type symmetry within the constant loading bytecodes. The choice to provide specialized load and store bytecodes for those slots that are frequently occupied by method parameters also negatively impacts the execution frequency of those specialized bytecodes. Consequently, we recommend that researchers modifying the Java Virtual Machine instruction set or developing new intermediate representations develop an asymmetric instruction set which provides more specialized bytecodes for those data types that are utilized with greatest frequency.

References

[1] T. Lindholm and F. Yellin. *The Java Virtual Machine Specification, 2nd Edition*. Addison-Wesley Publishing Company, Reading, Massachusetts, 1999.

[2] B. Stephenson and W. Holst. A quantitative analysis of the performance impact of specialized bytecodes in Java. In *CASCON '04: Proceedings of the 2004 conference of the Centre for Advanced Studies on Collaborative research*, pages 267–281. IBM Press, 2004.