An Introduction to Multicodes

Ben Stephenson
Department of Computer Science
University of Western Ontario
ben@csd.uwo.ca
Outline

• Java Virtual Machine Background
• The Current State of the Multicode Art
• Future Research Directions
• Getting More Information
The Java Virtual Machine

• A piece of software used to execute Java applications

• Java applications are made up of class files
  – Initialized Data (Constant Pool)
  – Field Information
  – Information for each Method
    • Bytecodes describing functionality of method
Java Bytecodes

• Similar to assembly language

• Java virtual machine is a stack based architecture
  – Math and logic bytecodes pop operands off stack and push result onto stack
  – Load and store bytecodes are used to access local variables
  – Get and put bytecodes are used to access fields
A Simple Method

```java
int i

void f()
{
    for (i = 0; i < 10; i++)
    {
        System.out.println(i);
    }
}
```
0  aload_0
1   iconst_0
2   putfield #5 <Field int i>
5   goto  28
8   getstatic #6 <Field java.io.PrintStream out>
11  aload_0
12  getfield #5 <Field int i>
15  invokevirtual #7 <Method void println(int)>
18  aload_0
19   dup
20  getfield #5 <Field int i>
23   iconst_1
24    iadd
25  putfield #5 <Field int i>
28  aload_0
29  getfield #5 <Field int i>
32    bipush 10
34    if_icmplt  8
37    return
0  aload_0
1   iconst_0
2  putfield #5 <Field int i>  \(i = 0\)
5   goto 28
8  getstatic #6 <Field java.io.PrintStream out>
11  aload_0
12  getfield #5 <Field int i>  \(System.out.println(i)\)
15 invokevirtual #7 <Method void println(int)>
18  aload_0
19   dup
20  getfield #5 <Field int i>  \(i++\)
23   iconst_1
24   iadd
25  putfield #5 <Field int i>
28  aload_0
29  getfield #5 <Field int i>  \(i < 10\)
32  bipush 10
34   if_icmplt 8
37  return
Executing a Java Method

• Execution starts at the beginning of the method

• A loop inside the VM is used to
  – Fetch the next bytecode
  – Transfer control to the correct place in the JVM to execute the bytecode
  – Execute the bytecode
  – Return to the top of the loop
Executing a Java Method

• Notice that there is a large amount of overhead!

• A loop is used to
  – Fetch the next bytecode
  – Transfer control to the correct place in the JVM to execute the bytecode
  – Execute the bytecode
  – Return to the top of the loop
Multicodes

- Multicodes reduce the amount of overhead by grouping several bytecodes together forming a new bytecode
- This reduces the number of times the loop must execute, reducing the amount of overhead
- Consider converting the sequence `aload_0 dup getfield iconst_1 iadd putfield` to a multicode named `fieldinc`
0 aload_0
1  iconst_0
2  putfield  #5  <Field  int  i>
5  goto   28
8  getstatic  #6  <Field  java.io.PrintStream  out>
11  aload_0  i = 0
12  getfield  #5  <Field  int  i>
15  invokevirtual  #7  <Method  void  println(int)>
18  fieldinc  #5  <Field  int  i>  i++
22  aload_0
23  getfield  #5  <Field  int  i>  i < 10
26  bipush  10
28  if_icmplt  8
29  return

- Five fewer bytecodes
- Five fewer fetch, transfer and loop operations per iteration of the loop
Identifying Bytecode Sequences

• We don’t want to do a static analysis of the class files
  – Does not consider code that is executed more than 1 time
  – Considers code that is never executed

• A dynamic analysis was performed
  – Every bytecode executed was recorded
  – The trace of the bytecodes was analyzed
Bytecode List for Heap Sort

1. `aload_1 iload_1 iload_3 daload dastore iload_3 iload_3 dup istore_3 goto`
2. `aload_1 iload_3 daload aload_1 iload_3 icnst_1 iadd daload dcmpg ifge`
3. `aload_0 getfield aload_0 getfield`
4. `dload aload_1 iload_3 daload dcmpg ifge`
5. `aload_0 dup getfield icnst_1 iadd putfield S3 arraylength if_icmpge`
6. `iload aload_0 getfield if_icmplt`
7. `S3 iload iadd caload iload_3 if_icmpne`
8. `iload_3 iload if_icmple`
9. `iload_1 i2c istore_3 iload_2 ifge`
10. `S3 aload_0 getfield caload invokevirtual`
Performance Results
(Multicodes of Length 2)
Performance Results
(Multicodes of lengths 2 through 5)
Performance Anomaly
(Multicodes of lengths 2 through 5)
Future Work

• Optimizing Multicodes
  – At the present time our multicodes are implemented by concatenating the functionality of the bytecodes.
  – This sequence of instructions can be optimized
  – Consider `iconst_1 iadd`

\[
\begin{align*}
top &++ \\
st[top] & = 1 \\
top &-- \\
st[top] & = st[top] + st[top+1]
\end{align*}
\]
Future Work

- Implement more multicodes
- Loosen the restrictions on multicodes
  - Allow multicodes containing branch bytecodes
- Perform substitutions at runtime
- Use multicodes in conjunction with a JIT
Getting More Information…


• Talk to me…
Questions?