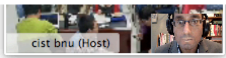
1. (easy) Study this table of compiler characteristics:



name	compile time for n bytecodes	execution time for generated code
interpreter	0	t
C1 compiler	$5n$	$0.25t$

and also study this table of method characteristics:

method	number of bytecodes	interpreter execution time, per call
`foo()`	30	100
`bar()`	50	120
`baz()`	1000	500

Suppose each method has been executed 10 times so far.

(a) 2 points Which of the three methods is the *best* candidate for recompilation?

(b) 3 points Which of the candidates should *not* be recompiled? Why not?

Method foo()
COST to compile is 5*30 == 150 + IMPROVED EXE TIME == 10*25== 250 (400)
10 calls of interpreted foo() is 10 * 100 == 1000

Method bar()
COST to compile is 5*50 = 250 + IMPROVED EXE TIME == 10*30 == 300 (550)
10 calls of interpreted bar() is 10 * 120 == 1200

Method baz()
COST to compile is 5*1000 == 5000 + IMPROVED EXE TIME == 10*125== 1250 (6250)
10 calls of interpreted baz() is 10 * 500 == 5000

a) method bar() would save the most time when recompiled, i.e. 1200 – 550
b) method baz() should not be recompiled, since the cost of recompilation is greater than the cost of continued interpretive execution.

Q3. (moderate) In a multi-core system, how might we reduce the observable pause time due to JIT compilation?

Do the JIT compilation in a background thread, and continue executing the user code (in the interpreter) in another thread

Q4. (moderate) Why might a hot method become cold during program execution?

Because sometimes past behaviour is NOT a predictor of future behaviour. Programs can go through **phase changes**. Another reason for hot methods going cold might be a change in the **environment.**

Q5. (difficult) In this presentation, we have used the method as the unit of re- compilation. Which other program structures might be suitable for recompilation?

syntactic structure – loop body. Inner bodies of loops might be sent to FPGAs or OpenCL GPUs for execution.

dynamic structure – program *trace – a linear sequence of instructions.* ***PyPy***

QUESTIONS FROM THE CLASS.

1、How does the compilation level effect execution time and generation time？

more optimization makes the compiler slow [COST], but the compiled code is fast [BENEFIT]

2、Why we always assume we are half-way through the program execution？

This is a heuristic. (guess, or approximation). Past behaviour predicts future behaviour.

Annotate the platform-neutral bytecode – to indicate which methods are hot (before execution).

```
@HotMethod
public void f() { …}
```

Can we use some tools or methods to make sure how many instructions we have executed？

Yes – profiling techniques. Method counters. (instrumentation).

```
public void f() {
        counters[F_ID]++;    // this code is automatically inserted


}
```

3、The differences between just-in-time compilers and interpreters.

most important

Interpreter goes through the bytecode linearly, and performs operations for each bytecode instruction. - FAST

JIT compiler takes a method at a time, and generates a block of native machine instructions for that method. - SLOW

4、How to understand the 'Pareto Principle¡¯with compilation？

Some parts of your code are executed more frequently than other parts of your code.

Pseudo-code

init()

read_in_a_file() – loops over the lines of the file

data_processing() – does some complex calculations in a loop

shutdown()

90% of the time is spent in 10% of the code

5、Could you tell some advantages about JIT? Do some explains.

focusing the compilation effort – only on hot code => don't waste compilation resource on code which doesn't need it.

deferring compilation till runtime (user execution)

- generate specialized code for the user's platform
- fitted for local processor extensions

- there is a single canonical version of the program (.jar archive, .pe)

6、Do static languages are easier to achieve JIT than dynamic languages?

yes. Static language – static type system (e.g. Java, Scala).

Dynamic language – dynamically typed – (e.g. Javascript, Python).

‘dynamic’ means we can dynamically generate and execute code inside the program. eval(string); - execute arbitrary code.