JIT Compilers Compilers Programming Languages +1

If a JIT compiler compiles the bytecode into machine code, what is the interpreter doing? Does it simply run the machine code? Do JIT and the interpreter work in parallel?

Answer Interesting · 6 Request

Paul Pacheco, works at American Airlines

Updated Sep 27, 2017

It is redundant. Bytecode that is compiled with the JIT to native code will no longer need the interpreter. Code that is interpreted does not go through the JIT.

Think of the interpreter as a big switch statement that receives the bytecodes for a function and does what the bytecode tells it to do.

```
void interpret(function) {
    foreach (instruction in function) {
        switch (instruction) {
            case ADD:
                // add the parameters
                break;
            case MULT:
                // multiply the parameters
                break;
            ...
        }
    }
}
```

So why have both at the same time? Good question, the reason is performance.

Suppose you need to run a java function once. How long would it take? Lets assume it takes 50ms to JIT compile a function, 1 ms to execute the JIT compiled function and 7 ms to interpret it:

Option 1, we JIT compile the function:

$$TotalTime = TimeToJIT + TimeToExecute = 50 + 1 = 51ms$$

Option 2, we Interpret the function:

$$TotalTime = TimetoInterpret = 7ms$$

Even though executing a JIT compiled function is faster than interpreting it, the time it takes to JIT compile means it's not worth it. Thus if we are running the

Related Questions

Is Java a compiled language or interpreted? What is the difference? What is the JIT compiler?

What are the differences between bytecode and machine code? Is bytecode specific to Java only?

How does the Java interpreter (JVM) convert bytecode into machine code?

Why are there interpreters (computer science) if compiling to machine code is faster?

In Java, what exactly will the JVM interpreter and the JIT compiler do with the bytecode?

Why do we call it "JIT compiler" and not "JIT interpreter" to refer to the thing that converts the

Is a virtual machine simply a compiler?

How do JIT compilers work?

How can an interpreter "directly execute" bytecode without transforming it into machine code?

The JVM interprets bytecode. In general, is every interpreter a virtual machine?

+ Ask New Question

More Related Questions

Question Stats

6 Publicly Interested

3,024 Views

Last Asked May 10, 2015

Edits

Option 1, we JIT compile the function:

$$TotalTime = TimeToJIT + TimeToExecute * 1000 = 50 + 1 * 1000 = 1050ms$$

Option 2, we interpret the function:

$$TotalTime = TimetoInterpret * 1000 = 7 * 1000 = 7000ms$$

Notice that we only need to JIT compile the function once and execute the compiled function 1000 times. In this case the time it takes to JIT compile the function dwarfs in comparison to the time it takes to execute it. The fastest way would be to JIT compile the function.

There are memory implications too. It takes more memory to JIT compile a function.

The VM is always playing a game of heuristics to determine when it is appropriate to interpret and when it is appropriate to JIT compile a function. It is able to execute the program faster over all by having both mechanisms available and choosing function by function whether to JIT or interpret it.

Eliot Miranda, Author of the Cog and BrouHaHa Smalltalk VMs. Ex tech lead for VisualWorks Smalltalk ('99 - '06).

Answered May 5

That's a great question. Some would say the interpreter is redundant but I disagree. I've worked on two Smalltalk VMs containing machine-code JITs. The first is HPS, the second generation JIT done by Peter Deutsch, Allan Schiffman, Frank Jackson and David Ungar. This is a pure JIT and has no interpreter at all. The second one is Cog, an extension of the Back-to-the-Future Squeak Smalltalk VM by Dan Ingalls et al. I architected the initial version of Cog in the light of my experiences with HPS, and I chose to keep the interpreter, for good reasons.

One important reason is start-up time. It is common for systems to run large initialization methods at start-up. JITing a method which is only executed once is a mistake. JIT compilation is akin to interpretation in that a pass is made over the bytecode of a method, converting it to machine-code, so at least the front end of a JIT for a byte coded language is a single-pass interpreter over the byte codes. This interpretation step along with the code generation phases mean that