Review:
A brief history of Just-In-time

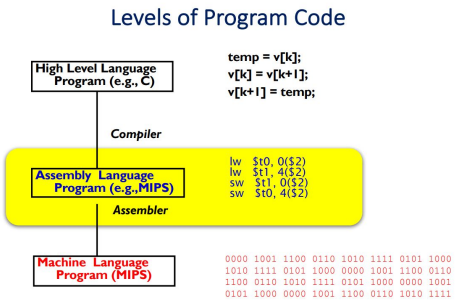
Xinhao Luo • 04.09.2021

View interactive version online with this link: https://docs.google.com/presentation/d/1Gi1cPtCRHP7CpiRlamLVZj8_KVNJLyH9a8uemsiiTQ0/edit

Overview

What is Just-In-Time, and Why?
What are the approaches?
Java and Just-In-Time
Outlook

Translation: bridge between human and machine



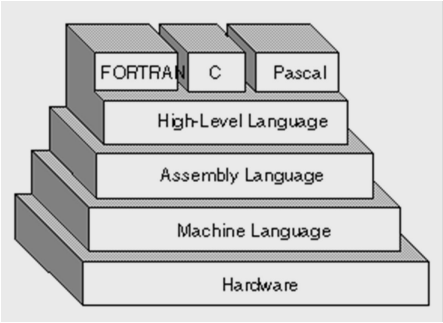
Compilation

- Involves at least two languages: source language and target language
- Higher level to lower/native language

Interpretation

- Read codes and execute block by block
- Won't convert all codes at once

Image source: https://raw.githubusercontent.com/colbybanbury/assemblyPicture/master/Screenshot%20from%202017-10-14%2014-03-06.png



Translation: bridge between human and machine

Compilation

- 👍 Run faster
- 👎 Platform specific
- 👎 Larger size

Interpretation

- 👍 Small in size
- 👍 Portable
- 👎 Poor performance

Image source: github.com

Just-In-Time: hello, world!

Origin

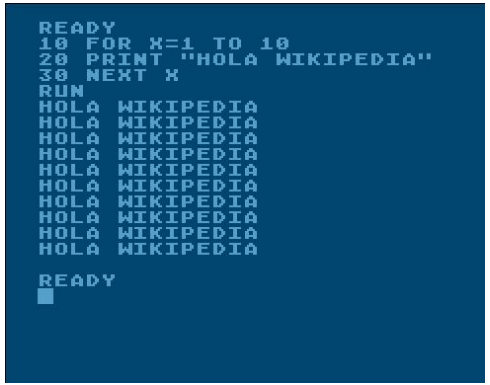
- University of Michigan Executive System for the IBM 7090 around 1975
- Possibility that assembler and loader could do translation during program execution

Ideas

- Compiled code could be extracted from the interpreter at runtime and used later
- Most of the programs are executing a small portion of their codes most of the time

IBM 7090, Image source: https://www.ibm.com/ibm/history/exhibits/mainframe/mainframe_PP7090.html

Just-In-Time: a tradeoff between speed and size, etc.



Mixed Code

- the program consists of native code and interpreted code
- The native code parts are frequently used while interpreted codes are seldom to be executed.

Throw-away Compiling

- Codes will be compiled dynamically when needed
- If the machine is under memory pressure, compiled code will be abandoned to free up space.

Basic language, Image source: https://en.wikipedia.org/wiki/BASIC

Just-In-Time: a tradeoff between speed and size, etc.

Mixed Code

- 👎 keep identical behavior across the compiler and the interpreter
- 👎 (optional) using compiler and interpreter at runtime is costly

Throw-away Compiling

- 👎 RAM--;
- (If you have enough memory, then it actually works pretty well)

Ram, image source: facebook.com/xataka

Field Testing

What kind of codes should we compile?

Frequency-of-executio n counter: compiling and optimizing these "hot" code blocks

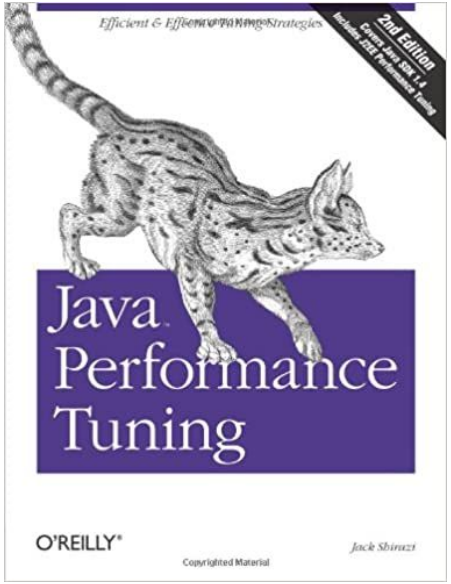
More sophisticated metrics e.g. use code profiles to reorder code blocks and improve hardware prediction in the background.

Introduced "opcodes", which injected translated codes directly into existing threads.

object-oriented style usually brought short methods it is better to optimize method callers by inlining these frequently called short methods

It asks users to explicitly annotate codes that needed to call just-in-time compiler

Java, JVM and Performance



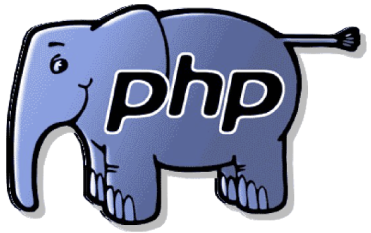
Observation

- People found that the performance of JVM was astonishingly bad, while it was really popular
- Cramer (one of the creator) and his team observed that 68% of the time was spent on interpreting the code

Focus

- Instead of focusing on the JVM performance only, it is found that optimizing Java code was also important
- Looking for a balance between the efficiency of the compile algorithm and the speed of java bytecode execution

Image source: https://www.amazon.com/Java-Performance-Tuning-Jack-Shirazi/dp/0596003773



Outlook(s)

1. JIT will play an important role in the near future
2. The ideas from the past still shined today
3. A good topic to explore further