IDPA: Programmiersprachen benchmarken

Nick Zbinden und Matthias Gasser

23. Februar 2011

1 Vorwort

Die Informatik hat auf uns schon immer eine gewisse Faszination ausgeübt. Es war daher für uns beide naheliegend eine Lehre als Informatiker zu machen. Während Nick vor allem an der Softwareentwicklung interessiert ist, bevorzuge ich die Systemtechnik, in der die Installation und Wartung von IT-Systemen der Grosse Schwerpunkt ist. Da wir beide unsere Berufe noch immer sehr interessiert ausführen, war es für uns beide klar für die IDPA ein Thema aus dem Bereich Informatik zu wählen. Es galt nun ein Thema zu finden, in dem sowohl die Applikationsentwicklung wie auch die Systemtechnik gleichermassen eine Rolle spielen. Bereits nach kurzer Recherche stiessen wir auf das Thema „Programmiersprachen Benchmarking“. Sofort war unser beider Interesse geweckt und dadurch, dass dieses Thema sowohl ein Testsystem wie auch selbstgeschriebene Programme erforderte, eignete es sich sehr gut für eine IDPA. Wir entschieden uns, unsere Maturaarbeit diesem Thema zu widmen. Am gewählten Thema interessiert uns beide in erster Linie ob tatsächlich Performanceunterschiede zwischen den untersuchten Sprachen feststellbar sind, und in welchem Ausmass sie sich zeigen. Heutige Computer sind sehr leistungsfähig, dadurch ist es für kleinere Programme wie wir sie testen werden meist irrelevant ob eine Programmiersprache einige Millisekunden schneller ist als die andere. Für grössere Projekte spielt die Geschwindigkeit jedoch auch heute noch eine wichtige Rolle, wir hoffen daher, dass sich unsere Testergebnisse auf grössere Applikationen projizieren lassen.

2 Abstract

Wir haben uns entschieden für unsere Maturaarbeit die Programmiersprachen Java, Clojure und Scala zu vergleichen. Wir haben dazu ein identisches Programm in jeder der genannten Programmiersprachen geschrieben. Durch das Messen von Prozessor- und Speicherdaten sowie der Ausführungszeiten der Programme wollten wir Unterschiede feststellen, um am Ende aufzeigen zu können, welche der getesteten Sprachen vom Testsystem am wenigsten Leistung abverlangt. Für die Messungen haben wir ein

Script geschrieben, welches die erforderlichen Daten in angemessenen Zeitabständen ausliest.

3 Einleitung

3.1 Untersuchungsgegenstand

Wir haben uns ein sehr komplexes Thema vorgenommen. Programmiersprachen und Compiler sind, auf die Informatik bezogen, einen sehr altes Thema.
Hochsprachen wie wir Sie in dieser Arbeit verwenden, werden seit mehr als fünfzig Jahren immer wieder weiter entwickelt und verbessert. Seit der Anfangszeit herrscht der Konflikt zwischen hoher Abstraktion und Geschwindigkeit. Umso höher die Abstraktion ist, die eine Programmiersprache bietet, umso schwieriger ist die Programmiersprache auf der darunterliegenden Sprache effizient auszuführen.

3.2 Problemstellung

Wir haben uns die Aufgabe gestellt drei Programmiersprachen unter dem Aspekt der Geschwindigkeit anzuschauen und zu vergleichen. Eine definitives Ergebnis ist in diesem Bereich fast unmöglich da die Anwendungsgebiete einer Programmiersprache zu verschieden sind und jeder Anwendungsfall andere Anforderungen hat. Auch die unter der Programmiersprache liegenden Layers (Betriebssysteme und Hardware) sind von entscheidender Bedeutung. Um korrekte Aussagen zu machen müssen diese Layers entweder herausgerechnet werden oder identisch sein. Das Ziel ist es für unseren Anwendungsfall Messungen zu machen, Aussagen über die Geschwindigkeiten zu treffen und wenn möglich klären warum die Sprachen in diesem Beispiel so schnell sind wie sie sind.

3.3 Wissenslücken

Um komplizierte Algorithmen in drei verschiedenen Sprachen zu programmieren braucht man viel Erfahrung in diesen Sprachen. Um den Aufwand nicht zu hoch werden zu lassen haben wir uns auf die Implementierung in einer Programmiersprache spezialisiert und vergleichen diese mit Referenz-Implementationen die wir nur erklären und messen.

4 Material und Methoden

4.1 Allgemeines zu den Versuchen

4.2 Verwendete Programmiersprachen

4.3 Das Testsystem

4.4 Aufnahme der Daten

4.5 Datenauswertungsmethoden

5 Ergebnisse

5.1 Allgemeines zu den Sprachen

Um es einfacher, fair und simpel zu machen haben wir uns entschieden Programmiersprachen zu wählen die auf der JVM (oder anderen VMs die Java Byte Code ausführen) laufen. Das Modell der VM bietet verschiedene Vorteile die ich kurz erläutern möchte.
Da VMs einen Bytecode als Input erhalten ist es grundsätzlich möglich jede Programmiersprache auf einer VM laufen zu lassen. Wie dieser Bytecode in der VM ausgeführt wird ist den darüber liegenden Sprachen egal.
Einige Möglichkeiten wie eine VM den Bytecode ausführt:

- Interpreter
- JIT-Compiler
- Native Code Compiler
- Direkt auf Hardware

Ich möchte nur auf den JIT-Compiler etwas näher eingehen da die JVM die wir benutzen einen solchen verwendet (Hotspot). Um zu verstehen warum ein Programm langsam oder schnell ist muss man bis zu einem gewissen grad den Compiler verstehen.

5.1.1 JIT Compiler

Ein JIT-Compiler kompiliert nicht alles auf einmal sondern, nur den Code der auch wirklich gebraucht wird. Das erlaubt es dem Compiler Optimierungen an den wichtigen Stellen anzubringen. Ein weiterer grosser Vorteil ist es, dass dem JIT-Compiler die Umgebung auf der er sich befindet bekannt ist. Das erlaubt es Optimierungen anzubringen die speziell für diese Hardware den Code anpassen.

Gute JIT-Compiler sind heutzutage vergleichbar mit der Geschwindigkeiten von Native Code Compilern. Die Unterschiede sind in vielen Anwendungsfällen nur noch gering.

5.2 Java

5.2.1 Beschreibung

Java ist eine Programmiersprache die ab 1992 von Sun Microsystems (oder teilweise im Auftrag von Sun) entwickelt wurde. Java wurde zuerst für eingebettete Systeme geschrieben. Heutzutage findet man Java Applikation in vielen Anwendungsgebieten. Java ist eine der meist verwendeten Programmiersprachen und wird auch in vielen Universitäten gelehrt.

5.3 Scala

5.3.1 Beschreibung

Die Entwicklung von Scala hat 2001 an der ETH Lausanne unter Martin Odersky begonnen. Die Idee war eine Sprache zu designen, welche die Konzepte von funktionalen und objektorientierten Sprachen in Synthese verwendet. Zwar wurde Scala im akademischen Kontext entwickelt findet aber immer mehr Anwendungen im Business Bereich.

5.4 Clojure

5.4.1 Beschreibung

Clojure ist eine dynamische Programmiersprache die 2007 von Rich Hicky für die JVM geschrieben wurde. Im Vordergrund der Entwicklung standen hohe Abstraktion, vor allem bei Concurency Programming, und eine gute Integration ins Host System (JVM). Diese erlaubt die Wiederverwendung von Java Code, Java Ecosystems (Webservers, Profilers, Debuggers...) und natürlich der VM.

6 Diskussion

6.1 Vergleich der Resultate

6.2 Schlussfolgerung

7 Abkürzungsverzeichnis

VM - Virtual Machine
JVM - Java Virtual Machine
IL - Intermediate Language

8 Literaturverzeichnis

9 Glossar

Virtual Machine: Eine Virtual Machine ist ein Programm, welches dem darauf ausgeführten Programm vorgibt, ein echter Computer zu sein. Sie bekommt als Eingabe eine beliebige Sprache und führt diese auf der darunter liegenden Sprache aus.

Java Virtual Machine: Eine Implementierung einer VM die dafür ausgelegt ist Java Bytecode auszuführen.

Bytecode: Ein Instruktionsset für eine VM.

Java Bytecode: Java Bytecode ist der für Java bzw. für die JVM entwickelte Bytecode.

10 Anhang