DHBW Mosbach
Lohrtalweg 10
74821 Mosbach
Deutschland

Ahead-of-Time-Compilation vs Profile-based Just in time Compilation

Seminar Informatik an der Dualen Hochschule Baden-Württemberg Mosbach

Studiengang/-richtung:	B.Sc. - Angewandte Informatik
Kurs:	INF20B
Name, Vorname:	Robin Wollenschläger
Matrikelnummer:	9495107
Name, Vorname des wiss. Prüfenden/Betreuenden:	Prof. Dr. A. Auch
Name und Sitz des Ausbildungsbetriebes:	GÖTTFERT Werkstoff-Prüfmaschinen GmbH
	Siemensstrasse 2, 74722 Buchen
Abgabedatum:	30. November 2022

Sperrvermerk

Der Inhalt dieser Arbeit darf weder als Ganzes noch in Auszügen Personen außerhalb des Prüfungsprozesses und des Evaluationsverfahrens zugänglich gemacht werden, sofern keine anders lautende Genehmigung der Ausbildungsstätte vorliegt.

Anmerkung Bilder/Screenshots

Bilder und Screenshots in dieser Arbeit wurden bewusst ohne Daten erstellt, um den datenschutzrechtlichen Bestimmungen zu entsprechen. Falls dies nicht möglich war, wurden Bereiche mit personenbezogenen Daten geschwärzt.

Ehrenwörtliche Erklärung

Ich versichere hiermit, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.
Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Buchen, 30. November 2022

Abkürzungsverzeichnis

PC Personal Computer

OS Operating System

JIT Just-In-Time

AOT Ahead-Of-Time

Abbildungsverzeichnis

Tabellenverzeichnis

Inhaltsverzeichnis

1. Abstract

1.1. Deutsch

1.2. English

2. Einleitung

2.1. Thema

Diese Arbeit befasst sich mit einem Vergleich der Implementierungsformen von Compilern Ahead-Of-Time (AOT) und Just-In-Time (JIT) anhand der historischen Entwicklung, den zugehörigen Programmiersprachen, sowie den Vor- und Nachteilen.
Über die historische Entwicklung wird Bezug zu Entstehungsgründen und technologischer Umsetzung, sowie Verbesserung genommen. Im Kontext der Implementierungsform JIT soll auch die Technologie des Profiling untersucht und erklärt werden, das zur Gewinnung von Performance genutzt wird.

2.2. Zielsetzung

Ziel dieser Arbeit ist die Ausarbeitung von Merkmalen und Unterschieden der beiden Implementierungsformen AOT-Compiler und JIT-Compiler, sowie deren Vorzügen in verschiedenen Anwendungsfällen und verschiedenen Sprachen.
Es soll dabei anhand der historischen Entwicklung auf Beweggründe für die Entstehung der einzelnen Techniken eingegangen werden, sowie beide unter verschiedenen Gesichtspunkten und Anwendungsfällen hinsichtlich Performance und Flexibilität verglichen werden. Anhand eines recherchierten Ergebnisses, soll eine Empfehlung für je nach Einsatzzweck für eine der genannten Implementierungsformen ausgesprochen werden.

2.3. Abgrenzung

Diese Arbeit soll kein direktes eigenes Experiment darstellen, dass einen klaren Sieger in einer fest definierten Disziplin hervorbringt, sondern ein breites Ergebnis anhand vielfältiger anderer Arbeiten präsentieren. Diese Ergebnisse anderer Arbeiten gilt es zu interpretieren und auf Basis dieser Schlussfolgerungen Empfehlungen auszusprechen.

3. Historische Grundlage

3.1. Implementierungsformen von Software

Um Programme in einer bestimmten Programmiersprache für einen Personal Computer (PC) lesbar zu machen müssen diese in Maschinencode, quasi die Muttersprache des PC's, übersetzt werden [3, vgl. Wagenknecht Hielscher 2022, S.2ff]. Dazu gibt es verschiedene technische Implementierungsformen, die folgend kurz vorgestellt werden:

3.1.1. Compiler

Der Compiliervorgang ist ähnlich einem Übersetzungsvorgang von z.B. einem chinesischen Text in englische Sprache zu sehen und kann gerade bei größeren Programmen durchaus Zeit in Anspruch nehmen und der Quellcode muss für jede Änderung erneut compiliert werden. Als Vorteil ist jedoch die schnelle Ausführung durch das Zielsystem zu nennen, da der Programmcode komplett in die korrespondierende Instruktionen dieses Zielsystems übersetzt wurde. [3, vgl. Wagenknecht Hielscher, S.121] Andererseits muss das Zielsystem bereits beim Compilationsvorgang bekannt sein und es kann nur allgemein für ein System, aber nicht zwangsläufig für die tatsächlich darunter liegende Hardware übersetzt werden.
Durch den Einsatz von höheren Programmiersprachen wurde der Einsatz von Compilern nötig, um den verfassten Quellcode für die jeweiligen Zielsysteme in den geeigneten Maschinencode zu übersetzen. Die für den Menschen einfacher zu verstehenden Programmiersprachen (auch Hochsprachen) bieten die Möglichkeit ihre Logik für verschiedene Zielsysteme durch einen Compiler zu übersetzen und somit sowohl dem Menschen, als auch dem PC ein gut lesbares Format vorzuhalten, [3, vgl. Wagenknecht Hielscher, S.1ff].
Bereits Konrad Zuse setzte für seinen Zuse Z4 auf eine Art höhere Sprache, das sogenannte Plankalkül und wollte mit einem "Planfertigungsgerätäutomatisch von einem mathematischen Rechenplan einen Lochstreifen mit gestanztem Maschinenplan anfertigen, [2, vgl. Heilige 2013, S.45]. Nach heutiger Terminologie ist dieses Planfertigungsgerät bereits als Compiler zu betrachten.
Der erste Compiler A-0 wurde von Grace Hopper 1952 vorgestellt und erleichterte den Prozess der Programmierung hinlänglich, da statt direkter Maschinenbefehle nun Code aus verständlichen Algorithmen, die in einem Katalog zusammengefasst waren, aufgebaut werden konnten. Diese Bausteine wurden von A-0 aus dem Katalog abgerufen und der Code in korrekter Reihenfolge zusammen gesetzt. Benötigte Speicherbereiche wurden automatisch reserviert und die zugehörige Adressierung ebenfalls vom Compiler übernommen, [1, vgl.

Beyer 2012, S.222ff]. In der Weiterentwicklung, dem Prototypen B-0 konnten Befehle in verschiedenen menschlichen Sprachen (z.B. Englisch, Französisch, Deutsch) verwendet werden, die der Computer zu Maschinencode übersetzte, [1, vg. Beyer 2012, S.271ff].
Über Compiler für die Sprachen Fortran und COBOL wurden die Compiler schlussendlich soweit entwickelt, wie wir sie heute kennen.

3.1.2. Interpreter

Ein Interpreter führt portionsweise Aktionen aus und interpretiert so zur Laufzeit den Quellcode zu maschinenlesbaren Instruktionen, wobei er im Unterschied zum Compiler keinen neuen Text erzeugt, sondern direkt den Quellcode zu Anweisungen übersetzt, [3, vgl. Wagenknecht Hielscher 2022, S.4]. Interpreter werden oft von Skriptsprachen eingesetzt und ersparen den Compilationsvorgang im Vorfeld.
Außerdem können Interpretersprachen plattformunabhängig ausgeliefert werden, da die Interpretation erst auf dem Zielsystem ausgeführt wird und im Vorfeld nicht bekannt sein muss, auf welchem Operating System (OS) der Quellcode ausgeführt werden soll. Das Zielsystem kann den Code somit für sich selbst interpretiere und die Instruktionen somit auch direkt für die eigene Hardware zielgerichtet umsetzen.
Andererseits entsteht durch die Interpretation zur Laufzeit ein erhöhter Aufwand, der Interpretersprachen grundsätzlich eine langsamere Ausführung beschert, als im Vorfeld zu Maschinencode kompilierten Programmen.

3.1.3. Just-in-time-Kompilierung

Die Implementierungsform der JIT-Kompilierung versucht die Vorteile von Unterabschnitt 3.1.2 und Unterabschnitt 3.1.1 zu vereinen. Einerseits wird der Code vorab zu Bytecode kompiliert, um eine schnellere Ausführung in einer virtuellen Maschine zu ermöglichen, andererseits wird der Code zur Laufzeit direkt auf der Zielmaschine von Bytecode in Maschinencode kompiliert, was den Vorteil bietet, dass dieser speziell für den ausführenden Prozessor optimiert werden kann.
JIT vereint damit den Vorteil der Plattformunabhängigkeit des Interpreters mit dem Geschwindigkeitsvorteil des Compilers. Die Vorteile können dabei nicht genauso stark ausgespielt werden, wie bei den ursprünglichen Implementierungsformen, aber es wird ein sehr guter Mittelweg geschaffen, um sich die Leistung moderner Rechner zunutze zu machen und direkt auf dem Zielsystem zu compilieren, um die Geschwindigkeit des ausgeführten Programmteils maßgeblich zu erhöhen, als auch die Auslieferung von direkt übersetzbaren Bytecode, der die Software plattformunabhängig macht.

3.2. Programmausführung in der virtuellen Maschine

Durch die Dynamik einer Programmiersprache, zur Schaffung einer Plattformunabhängigkeit oder aus Sicherheitsgründen werden Programme heutzutage oftmals nicht mehr direkt vom Betriebssystem ausgeführt, sondern laufen in einer virtuellen Maschine. Dies verzögert jedoch die Ausführungszeit der Interpretation der Befehle.
Durch die Anwendung von JIT wird hier versucht Performance zu gewinnen und somit diesen Nachteil auszugleichen.

4. Vergleich AOT vs. JIT

4.1.

5. Fazit

X. Anhang

Literatur

[1] Kurt W. Beyer. *Grace Hopper and the Invention of the Information Age* -. Cambridge: MIT Press, 2012. ISBN: 978-0-262-51726-3.

[2] Hans Dieter Hellige. *Geschichten der Informatik - Visionen, Paradigmen, Leitmotive*. Berlin Heidelberg New York: Springer-Verlag, 2013. ISBN: 978-3-642-18631-8.

[3] CHRISTIAN WAGENKNECHT und MICHAEL HIELSCHER. *Formale Sprachen, abstrakte Automaten und Compiler*. Bd. 3. Springer Vieweg, 2022.