ECOLE POLYTECHNIQUE FEDERALE DE LAUSANNE

MASTER THESIS

Efficient Deoptimization

Author:
Adrien GHOSN

Supervisor:
Prof. Jan VITEK, Prof. Viktor KUNCAK

A thesis submitted in fulfilment of the requirements for the degree of Master in Computer Science

in the

LARA
Computer Science

December 23, 2015

Declaration of Authorship

I, Adrien GHOSN, declare that this thesis titled, "Efficient Deoptimization" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

"Thanks to my solid academic training, today I can write hundreds of words on virtually any topic without possessing a shred of information, which is how I got a good job in journalism."

Dave Barry

ECOLE POLYTECHNIQUE FEDERALE DE LAUSANNE

Abstract

Faculty Name
Computer Science

Master in Computer Science

Efficient Deoptimization

by Adrien GHOSN

The Thesis Abstract is written here (and usually kept to just this page). The page is kept centered vertically so can expand into the blank space above the title too...

Acknowledgements

The acknowledgements and the people to thank go here, don't forget to include your project advisor…

Contents

List of Figures

List of Tables

List of Abbreviations

List of Symbols

ω	angular frequency	rad

For/Dedicated to/To my…

Chapter 1

Introduction

Chapter 2

Related Work

2.1 On Stack Replacement, General Principle

2.1.1 Definition & Overview

On-Stack replacement (OSR) is a set of techniques that consist in dynamically transferring the execution, at run time, between different pieces of code. The action of transferring the execution to another code artefact is called an OSR transition.

On-Stack replacement can be viewed, at a high level, as a mechanism that allows to transform the currently executing code, into another version of itself. This transformation mechanism has been used to allow the bi-directional transition between different levels of code optimisations. We can therefore reduce it to two main purposes: transforming an executing piece of code into a more optimised version of itself, and undoing transformations that were previously performed. While similar, these two types of transformation have very different goals.

In several virtual machines (CITE PAPERS), some of which will be presented in (REFERENCE), On-Stack replacement has been used to improve the performance of long running functions. When the VM identifies a piece of code as being "hot", i.e., it hogs the execution, it suspends its execution, recompiles it to a higher level of optimisation, and transfers the execution to the newly generated version of the function. This differs from a simple Just-In-Time (JIT) compiler, since the recompilation takes place during the execution of the function, rather than just before its execution. However, both techniques rely on run time profiling data to uncover new optimisation opportunities.
In this case, OSR is used to improve performance.

On-Stack replace allows to

Chapter 3

Theoretical Model

3.1 The OSR points

3.2 The Transition Mechanism

3.3 Constaints

Chapter 4

Implementation

Appendix A

Appendix Title Here

Write your Appendix content here.