

Bachelor Thesis

# Profile Caching for the Java Virtual Machine

Marcel Mohler  
ETH Zurich

Zoltán Majó  
Tobias Hartmann  
Responsible engineers

Prof. Thomas R. Gross  
Laboratory for Software Technology  
ETH Zurich

August 2015



Eidgenössische Technische Hochschule Zürich  
Swiss Federal Institute of Technology Zurich



Laboratory for Software Technology



# Introduction

Virtual machines (VMs) like the Java Virtual Machine (JVM) are used as the execution environment of choice for many modern programming languages. VMs interpret a suitable intermediate language (e.g., Java Byte Code for the JVM) and provide the runtime system for application programs. VMs usually include a garbage collector, a thread scheduler, and interfaces to the host operating system. As interpretation of intermediate code is time-consuming, VMs usually include a *Just-in-time* (JIT) compiler that translates frequently-executed functions or methods to *native* machine code.

The JIT compiler executes in parallel to a program's interpretation by the VM and, as a result, compilation speed is a critical issue in the design of a JIT compiler. Unfortunately, it is difficult to design a compiler such that the compiler produces good or excellent code while limiting the resource demands of this compiler. The compiler requires storage, CPU cycles and even on a multi-core processor, compilation may slow down the execution of the application program.

Consequently, most VMs adopt a multi-tier compilation system. At program startup, all methods are interpreted by the virtual machine (execution at Tier 0). The interpreter gathers execution statistics called *profiles* and if a method is determined to be executed frequently, this method is then compiled by the Tier 1 compiler. Methods compiled to Tier 1 are then profiled further and based on these profiling information, some methods are eventually compiled at higher tiers. One of the drawbacks of this setup is that for all programs, all methods start in Tier 0, with interpretation and profiling by the VM. However, for many programs the set of the most used methods does not change from one execution to another and there is no reason to gather profiling information again.

The main idea of this thesis is to cache these profiles from a prior execution to be used in further runs of the same program. Having these *cached profiles* available avoids the JIT compiler to gather the same profiling information again. As well as allow the compiler to use more sophisticated profiles early in program execution and prevent recompilations when more information about the method is available. While this in general should not significantly influence the peak performance of the program, the hope is to decrease the time the JVM needs to achieve it, the so called *warmup*.

This thesis proposes a design and an implementation of a profile caching feature for *HotSpot*, an open source Java virtual machine maintained and distributed by Oracle Corporation as well as a profound performance analysis using state-of-the-art benchmarks.



# Contents

<b>1</b>	<b>Overview of HotSpot</b>	<b>1</b>
1.1	Tiered compilation . . . . .	1
1.2	Deoptimizations . . . . .	3
1.3	On-Stack Replacement . . . . .	4
1.4	Compile thresholds . . . . .	4
<b>2</b>	<b>Motivation</b>	<b>7</b>
2.1	Example 1 . . . . .	8
2.2	Example 2 . . . . .	9
2.3	Similar systems . . . . .	11
<b>3</b>	<b>Implementation / Design</b>	<b>13</b>
3.1	Creating cached profiles . . . . .	13
3.2	Initializing cached profiles . . . . .	15
3.3	Using cached profiles . . . . .	16
3.4	Different usage modes for cached profiles . . . . .	17
3.4.1	Compile Thresholds lowered (mode 0) . . . . .	18
3.4.2	Unmodified Compile Thresholds (mode 1) . . . . .	18
3.4.3	Modified C1 stage (mode 2) . . . . .	18
3.5	Issues . . . . .	19
3.6	Debug output . . . . .	19
<b>4</b>	<b>Performance</b>	<b>21</b>
4.1	Setup . . . . .	21
4.2	Startup performance . . . . .	21
4.3	Deoptimizations . . . . .	23
4.4	Effect on compile queue . . . . .	28
<b>5</b>	<b>Possible improvements</b>	<b>37</b>
<b>6</b>	<b>Conclusion</b>	<b>39</b>

<b>A</b>	<b>Appendix</b>	<b>41</b>
A.1	Tiered Compilation Thresholds . . . . .	42
A.2	Cached Profile Example . . . . .	43
A.3	Code Changes . . . . .	44
A.4	SPECjvm Benchmark . . . . .	44
A.5	Octane Benchmark . . . . .	46
	<b>Bibliography</b>	<b>48</b>