Bachelor Thesis

Profile Caching for the Java Virtual Machine

Marcel Mohler

Zoltán Majó
Tobias Hartmann
Responsible assistants

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

August 2015

Introduction

(Language) Virtual Machines like the Java Virtual Machine (JVM) are used as the execution environment of choice for many modern programming languages. The VMs interpret a suitable intermediate language (e.g., Java Byte Code for the JVM) and provide the runtime system for application programs and usually include a garbage collector, a thread scheduler, interfaces to the host operating system. As interpretation of intermediate code is time-consuming, VMs include usually a Just-in-Time (JIT) compiler that translates frequently-executed functions or methods to "native" code (e.g., x86 instructions). The JIT compiler executes in parallel to a program's interpretation by the VM, and as a re- sult, 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 and 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. The first tier is the interpretation of a method. If this method is "hot", the Tier-1 compiler translates this method into a native code. The Tier-1 compiler implements only a small set of the know optimization techniques and as result, it had good compilation speed but the generated code is far from the output of an optimizing compiler. Such a compiler is usually the Tier-2 compiler, which takes a longer amount of time and produces optimized native code. To determine which methods should be compiled by the Tier-1 (or Tier-2) compiler, the VM profiles the execution of all application programs to identify "hot" methods. In current VMs, at program startup, all methods are interpreted by the VM (execution at Tier- 0). The interpreter performs profiling, and if a method is determined to be "hot", this method is then compiled by the Tier-1 compiler (fast compilation, few optimizations). Methods compiled to Tier 1 are then profiled further (and more extensively). Based on that profiling information, some methods are eventually compiled at Tier 2 (slow compilation, many optimizations). 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 "hot" methods does not change from one execution to another and there is no reason to gather again and again the profiling information. Cached profiles would allow the VM to compile some methods right away.

Contents

1 Introduction

Explain scope and structure of report.

2 Great Work

This and the following chapters detail the original work.

3 Examples

This chapter provides some additional hints and examples for the layout and style of the thesis. It is worthwhile to look at the source file `Examples.tex` for this appendix to understand how it was created.

3.1 Tables

Tables are left justified and the caption appears on top as seen in Table 3.1.

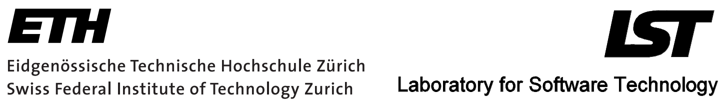
English	**German**
cell phone	Handy
Diet Coke	Coca Cola light

Table 3.1: Translations.

3.2 Figures

Figure 3.1 shows a simple figure with a single picture and Figure 3.2 shows a more complex figure containing subfigures.

Figure 3.1: LST logo.


ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich
LST
Laboratory for Software Technology

(a) ETH logo (b) LST logo

Figure 3.2: Two pictures as part of a single figure through the magic of the subfigure package.

Listing 3.1: Example usage of the listing package

```
class S {
   int f1 = 42;
   public S(int x) {
          f1 = x;
   }
}
```

3.3 Units

The SIUnits package provides nice spacing for units as demonstrated in Table 3.2. Use of the package also makes it easy to change the style or even the unit text in the future.

Output	**Command**
42m	`42m`
42 m	`\unit{42}{\metre}`
42 m	`42 m`

Table 3.2: Spacing for units.

3.4 Source code

The listings package provides tools to typeset source code listings. It supports many programming languages and provides a lot of formatting options.

Listing 3.1 shows an example listing. Code snippets can also be inserted in normal text: `\lstinline|int f1 = 42;|` gives `int f1 = 42;`

3.5 Miscellany

Capitalization. When referring to a named table (such as in the previous section), the word *table* is capitalized. The same is true for figures, chapters and sections.

Bibliography. Use `bibtex` to make your life easier and to produce consistently formatted entries.

Contractions. Avoid contractions. For instance, use “do not” rather than “don’t.”

Style guide. A classic reference book on writing style is Strunk’s *The Elements of Style* [**?**].

A Extra Stuff

Additional material such as long mathematical derivations.

Bibliography

[1] T. Hartmann, A. Noll, and T. R. Gross. Efficient code management for dynamic multi-tiered compilation systems. In *2014 International Conference on Principles and Practices of Programming on the Java Platform Virtual Machines, Languages and Tools, PPPJ '14, Cracow, Poland, September 23-26, 2014*, pages 51–62, 2014.