Using HPM-Sampling to Drive Dynamic Compilation

Dries Buytaert[†] Andy Georges[†] Michael Hind[*] Matthew Arnold[*] Lieven Eeckhout[†]
Koen De Bosschere[†]

[†] Department of Electronics and Information Systems, Ghent University, Belgium
[*] IBM T.J. Watson Research Center, New York, NY, USA
{dbuytaer,ageorges}@elis.ugent.be, {hindm,marnold}@us.ibm.com, {leeckhou, kdb}@elis.ugent.be

Abstract

All high-performance production JVMs employ an adaptive strategy for program execution. Methods are first executed unoptimized and then an online profiling mechanism is used to find a subset of methods that should be optimized during the same execution. This paper empirically evaluates the design space of several profilers for initiating dynamic compilation and shows that existing online profiling schemes suffer from several limitations. They provide an insufficient number of samples, are untimely, and have limited accuracy at determining the frequently executed methods. We describe and comprehensively evaluate HPM-sampling, a simple but effective profiling scheme for finding optimization candidates using hardware performance monitors (HPMs) that addresses the aforementioned limitations. We show that HPM-sampling is more accurate; has low overhead; and improves performance by 5.7% on average and up to 18.3% when compared to the default system in Jikes RVM, without changing the compiler.

Categories and Subject Descriptors D.3.4 [*Programming languages*]: Processors—Compilers; Optimization; Run-time environments

General Terms Measurement, Performance

Keywords Hardware Performance Monitors, Java, Just-in-time compilation, Profiling

1. Introduction

Many of today's commercial applications are written in dynamic, type-safe, object-oriented languages, such as Java, because of the increased productivity and robustness these languages provide. The dynamic semantics of such a language require a dynamic execution environment called a virtual machine (VM). To achieve high performance, production Java virtual machines contain at least two modes of execution: 1) *unoptimized* execution, using interpretation [21, 28, 18] or a simple dynamic compiler [16, 6, 10, 8] that produces code quickly, and 2) *optimized* execution using an optimizing dynamic compiler. Methods are first executed using the unoptimized execution strategy. An online profiling mechanism is used to find a subset of methods to optimize during the same execution. Many systems enhance this scheme to provide multiple levels of optimized execution [6, 18, 28], with increasing compilation cost and benefits at each level. A crucial component to this strategy is the ability to find the important methods for optimization in a low-overhead and accurate manner.

Two approaches that are commonly used to find optimization candidates are method invocation *counters* [10, 18, 21, 28] and timer-based *sampling* [6, 8, 18, 28, 30]. The counters approach counts the number of method invocations and, optionally, loop iterations. Timer-based sampling records the currently executing method at regular intervals using an operating system timer.

Although invocations counters can be used for profiling unoptimized code, their overhead makes them a poor choice for use in optimized code. As a result, VMs that use multiple levels of optimization rely exclusively on sampling for identifying optimized methods that need to be promoted to higher levels. Having an accurate sampler is critical to ensure that methods do not get stuck at their first level of optimization, or in unoptimized code if a sample-only approach is employed [6, 8].

Most VMs rely on an operating system timer interrupt to perform sampling, but this approach has a number of drawbacks. First, the minimum timer interrupt varies depending on the version of the OS, and in many cases can result in too few samples being taken. Second, the sample-taking mechanism is untimely and inaccurate because there is a delay between the timer going off and the sample being taken. Third, the minimum sample rate does not change when moving to newer, faster hardware; thus, the effective sample rate (rela-

OOPSLA'07, October 21–25, 2007, Montréal, Québec, Canada.
Copyright © 2007 ACM 978-1-59593-786-5/07/0010…$5.00