UNIVERSITY OF CALIFORNIA
Santa Barbara

Analysis, Detection, and Exploitation of Phase Behavior in Java Programs

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Priya Nagpurkar

Committee in Charge:

Professor Chandra Krintz, Chair

Professor Timothy Sherwood

Professor Tobias Hollerer

September 2007

UMI Number: 3283656

UMI Microform 3283656

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

The Dissertation of
Priya Nagpurkar is approved:

Professor Timothy Sherwood

Professor Tobias Hollerer

Professor Chandra Krintz, Committee Chairperson

August 2007

Analysis, Detection, and Exploitation of Phase Behavior in Java Programs

by

Priya Nagpurkar

Dedication and Gratitude

I dedicate this dissertation to my parents, Ashok and Rekha Nagpurkar, who never cease to amaze me with the perfect balance between independence and guidance that they have always struck in influencing my life. Their faith, encouragement and support, further reinforced by that of my brother, Ravindra, has played an important role in my decision to pursue, and in successfully completing this work.

My advisor, Chandra Krintz, played an equally important role, by recognizing and bringing out my potential for research. Her energy and enthusiasm for both research, and teaching were, and will continue to be, a constant inspiration. She has been a model *guru*, by being a good friend, philosopher, and guide. Many thanks also to Tim Sherwood, and Tobias Hoellerer for their guidance as members of my dissertation committee, and to our collaborators from the I.B.M. T.J. Watson Research Center for their valuable advice. Michael Hind, Peter Sweeney, Trey Cain, Mauricio Serrano, and Jong-Deok Choi were all excellent mentors.

I would like to express deep gratitude towards all my friends, old and new. Special thanks to my close friends, Rekha, Shilpa, Puja, Nicole, and Martina for being my extremely reliable support structure in times of need; to Lingli and Ye, for being great colleagues and neighbors; to Selim for great evenings in the climbing gym; and to Hussam for all the cups of tea; These and other members of the RACE lab made it a great place to work or to hang out. Visits to the CS office were always pleasant, thanks to Amanda, Greta, Julia, Beejay, and the rest of our very cheerful office staff. Finally, playing ultimate with the CS team was always something to look forward to – thanks to all of you on the ultimate team for your camaraderie!

Acknowledgements

The text of Chapter 3 is in part a reprint of the material as it appears in the proceedings of Elsevier Science of Computer Programming – Special Issue on Principles Practices and Programming in Java, Vol. 59. The dissertation author was the primary researcher and author and the co-author listed on this publication ([82]) directed and supervised the research which forms the basis for Chapter 3.

The text of Chapter 4 is in part a reprint of the material as it appears in the proceedings of the Fourth Annual International Symposium on Code Generation and Optimization (CGO). The dissertation author was the primary researcher and the co-authors listed on this publication ([80]) directed and supervised the research which forms the basis for Chapter 4.

The text of Chapter 6 is in part a reprint of the material as it appears in the proceedings of ACM Transactions on Architecture and Code Optimization (TACO),Vol. 3, Number 1. The dissertation author was the primary researcher and author with significant contribution from one of the co-authors, Hussam Mousa. The remaining co-authors listed on this publication ([84]) directed and supervised the research which forms the basis for Chapter 6.

The text of Chapter 7 is in part a reprint of the material as it appears in the proceedings of the Sixteenth International Conference on Parallel Architectures and Compilation Techniques (PACT). The dissertation author was the primary researcher and author and the co-authors listed on this publication ([79]) directed and supervised the research which forms the basis for Chapter 7.

Curriculum Vitæ

Priya Nagpurkar

Education

2007	**Doctor of Philosophy in Computer Science,** University of California, Santa Barbara.
2007	**Master of Science in Computer Science,** University of California, Santa Barbara.
2001	**Bachelor of Engineering in Computer Engineering,** Pune University.

Professional Experience

2006	**Summer Intern,** I.B.M. T.J. Watson Research Center.
2003 – 2007	**Graduate Research Assistant,** University of California, Santa Barbara.
2001 – 2003	**Graduate Teaching Assistant,** University of California, Santa Barbara.
2000	**Intern,** VERITAS Software India Ltd. (now Symantec), Pune, India.

Professional Activities

2007	**Program Committee Member**, International Conference on Principles and Practices of Programming in Java
2006	**Program Committee Member**, UCSB Graduate Student Research Conference
2006	**Submissions Chair**, International Conference on Principles and Practices of Programming in Java
2006-2007	**Graduate Student Representative**, Colloquium Committee
2005-2006	**Graduate Student Representative**, Graduate Admissions Committee

Publications

Priya Nagpurkar, Harold W. Cain, Mauricio Serrano, Jong-Deok Choi and Chandra Krintz: "Call-chain Software Instruction Prefetching in J2EE Server Applications," *In the Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT07)*

Lingli Zhang, Chandra Krintz, and Priya Nagpurkar: "Language and Virtual Machine Support for Efficient Fine-Grained Futures in Java," *In the Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT07)*

Lingli Zhang, Chandra Krintz, and Priya Nagpurkar: "Supporting Exception Handling for Futures in Java," *In the Proceedings of the International Conference on the Principles and Practice on Programming in Java (PPPJ07)*

Priya Nagpurkar, Harold W. Cain, Mauricio Serrano, Jong-Deok Choi and Chandra Krintz: "A Study of Instruction Cache Performance and the Potential for Instruction Prefetching in J2EE Server Applications," *Tenth Workshop on Computer Architecture Evaluation Using Commercial Workloads (CAECW-10)*

Priya Nagpurkar, Michael Hind, Chandra Krintz, Peter Sweeney, and V.T. Rajan: "Online Phase Detection Algorithms," *In the Proceedings of the International Symposium on Code Generation and Optimization (CGO06)*

Priya Nagpurkar, Chandra Krintz, and Timothy Sherwood: "Phase-aware Remote Profiling," *In the Proceedings of the International Symposium on Code Generation and Optimization (CGO05)*

Priya Nagpurkar and Chandra Krintz: "Visualization and Analysis of Phased Behavior in Java Programs," *In the Proceedings of the International Conference on the Principles and Practice of Programming in Java (PPPJ04)*

Priya Nagpurkar, Hussam Mousa, Chandra Krintz, and Timothy Sherwood: "Efficient Remote Profiling for Resource-Constrained Devices," *In the Proceedings of ACM Transactions on Architecture and Code Optimization (TACO),Vol. 3, Number 1, March, 2006, pages 1-32*

Priya Nagpurkar, and Chandra Krintz: "Phase-Based Visualization and Analysis of Java Programs," *In the Proceedings of Elsevier Science of Computer Programming – Special Issue on Principles Practices and Programming in Java, Vol. 59, Number 1-2, January 2006, pages 64-81*

Selim Gurun, Priya Nagpurkar and Ben Zhao: "Energy Consumption and Conservation in Mobile Peer-to-peer Systems," *The first International Workshop on Decentralized Resource Sharing in Mobile Computing and Networking (MobiShare06)*

Field of Study: Computer Science

Abstract

Analysis, Detection, and Exploitation of Phase Behavior in Java Programs

Priya Nagpurkar

The Java programming language offers developers many productivity enhancing features, including high-level abstractions, extensive libraries, architecture-independent execution, and type safety. These features are enabled by an intelligent execution environment that, incrementally and dynamically, compiles and executes compact representations of Java programs encoded for a virtual machine. While this necessarily adds overhead, the ability to compile (and recompile) code at runtime also enables the execution environment to perform dynamic, performance-enhancing optimizations based on the runtime behavior of the executing program. There are three primary steps in developing effective adaptive optimizations for these systems: (1) Development of a thorough analysis, understanding, and characterization of the performance of Java programs; (2) Extracting accurate data from programs efficiently at runtime; and (3) Guiding optimizations using feedback from the extracted performance data.

We address each of these steps in our research by focusing on techniques that capture and exploit the repeating patterns in program behavior (phases) within virtual execution environments, and in particular, those for Java programs. This dissertation can

be decomposed into two foci: phase analysis and detection tools and techniques and phase-aware techniques for efficient program analysis and optimization. We first study the time varying behavior of Java programs, show that Java programs do exhibit phase behavior, and present tools to extract and analyze this phase behavior. We then investigate the problem of accurate online phase detection for Java programs, within a Java virtual machine, the parameters that impact doing so effectively, and evaluate numerous online phase detectors. Finally we demonstrate the potential of phase-based optimizations by designing and evaluating two phase-based runtime techniques. The first technique is an accurate, low-overhead profiling scheme for resource-constrained devices that uses phases to drive when to sample the execution of a program. The second technique is a software instruction prefetching mechanism that uses method-level phase behavior to identify, predict, and prefetch methods that incur a large number of instruction cache misses for emerging Java workloads like database- and application servers. These two techniques span two extremes of execution environments used for Java applications: software for resource-constrained devices at the low end and application servers at the high-end.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The greatest happiness for the thinking person is to have explored the explorable and to venerate in equanimity that which cannot be explored.

Johann Wolfgang von Goethe (1749-1832)

The Java programming language, and similarly C# and the Microsoft .Net languages, offer many benefits to programmers such as portability, programmer productivity through high-level abstractions and extensive libraries, type and memory safety, and dynamic loading. These features make it easier, not only to develop software, but also to debug and maintain it. As a result, these languages are very popular with software developers. Java, in particular, has seen tremendous growth since its inception, a little over a decade ago, and the trend is predicted to continue [27]. It is estimated that Java today drives a $100 billion a year software industry, and is deployed on a wide variety of devices, including millions of desktops, billions of embedded devices (from smart phones to car navigation systems), and enterprise servers [101, 1].

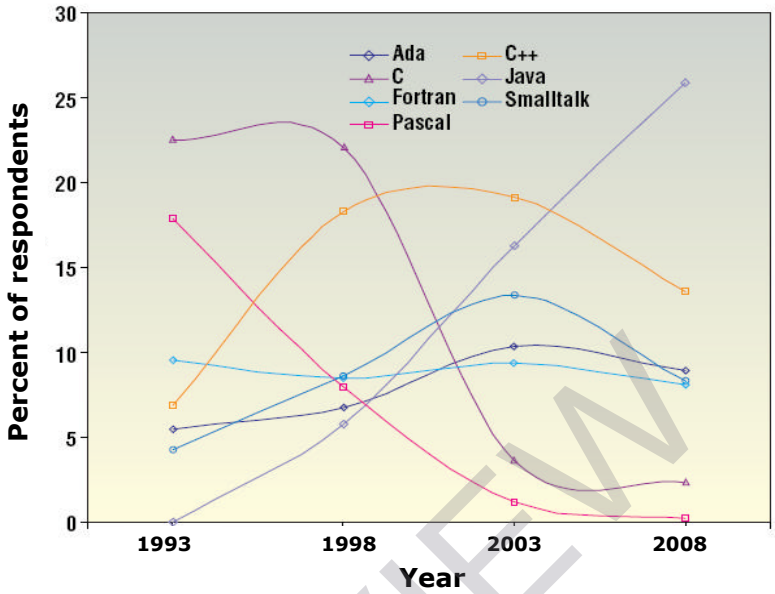

Figure 1.1: Programming Language Usage Trends. This figure from [27] uses historic data, gathered from real users, as well as prediction based on this data to show the long term trend in programming language popularity.

To enable these features, especially portability (the write-once, run-anywhere model), these programs are compiled into an architecture-independent intermediate format and executed within a virtual execution environment on the target host. The execution environment, a Java virtual machine or .Net runtime, implements a compilation system that converts the intermediate code to the native format of the underlying machine. This dynamic compilation necessarily introduces runtime overhead, but at the same time also exposes opportunities for adaptation – optimizations that we can customize according to the behavior of the executing program. State of the art Java Virtual Machines (JVMs) [71, 28, 102, 5, 53] employ adaptive optimization techniques based on

information gathered while the program is executing (feedback-directed optimization). Monitoring, analyzing, and predicting runtime program behavior are vital to feedback-directed optimization.

Recent research has shown, for non-Java programs, that program behavior varies over time, and exhibits repeating patterns [93, 41], and has focused on automatically characterizing this behavior [94, 95, 66, 35]. Phase analysis of programs is one such characterization, which isolates distinct behaviors in a program's execution by grouping periods of execution that are similar together in a *phase*. A program's execution can then be seen as a series of phases that might repeat themselves several times. Researchers have used phase behavior to improve program performance via hardware and software optimization [38, 96, 92, 72] and to reduce simulation time [94, 95] and generate cycle-close traces [88]. The ability to detect and predict phases at runtime has the potential of uncovering new opportunities in performing proactive adaptive optimizations for Java programs. With the ultimate aim of enabling better performance for Java programs, the thesis question that we explore is this work is:

> *How can we efficiently detect, track, predict, and exploit repeating patterns in program behavior (phases) in a Java virtual machine to facilitate runtime analysis and feedback-directed optimization in Java programs?*

To answer this question, we focus our efforts on phase characterization and detection in Java programs, and exploiting repetitions that phases manifest, using optimiza-

tion for a range of devices for which Java Virtual Machines are available. In particular, we

- investigate and develop offline mechanisms and tools that enable the characterization, visualization, and manipulation of phase behavior in Java programs,
- develop a modular, pluggable framework for implementing and investigating online phase detection algorithms within a Java Virtual Machine,
- devise and investigate a phase-aware approach to collecting accurate online execution profiles
- devise and investigate a prefetching scheme (a dynamic optimization) that exploits repeating patterns in Java server execution.

We take an empirical and implementation-oriented approach to developing phase detection and exploitation techniques in this thesis. We implement and empirically evaluate our techniques using a wide range of real programs and open-source Java virtual machine technologies. From this effort, we have produced a set of tools that significantly facilitate analysis of repeating patterns in the behavior of Java programs, and we have designed and evaluated novel techniques for exploiting phases to improve program profiling and execution performance.

The dissertation is organized as follows. We begin with a discussion of the relevant background in Chapter 2. This chapter includes techniques that we use in our work, as

well as the state of the art for phase analysis and its uses. We present our framework and toolkit for understanding and analyzing phase behavior in Java programs in Chapter 3, followed by our framework for the design and analysis of online phase detection algorithms within a JVM, in Chapter 4. Chapters 5, 6, and 7 focus on the use of phase behavior to enable two phase-based runtime techniques; Chapter 5 introduces these techniques and provides an overview, while the following Chapters provide details of each technique. We end with a summary of the dissertation and a discussion of future directions in Chapter 8.

Chapter 2

Background

The focus of this dissertation is understanding, analyzing, and exploiting the repeating patterns, or phases, in the time varying behavior of Java programs. Of particular interest to us, is the possibility of incorporating phase-awareness in virtual execution environments, like Java virtual machines, to drive adaptive, feedback-directed optimization of dynamically compiled programs.

Much research has already gone into the characterization, detection, prediction, and exploitation of phase behavior, especially in the area of computer architecture. In this Chapter, we present an overview of extant work on phase behavior, and also briefly describe the process of dynamic compilation for Java.

2.1 Phase Characterization, Detection, and Prediction Techniques

Program behavior has commonly been abstracted in the form of profiles gathered over the program's execution. Recently, there has been a lot of interest in studying program behavior during different parts of execution. Many researchers have observed, through detailed simulations and temporal profiles, that programs exhibit widely varying behavior during different parts of execution [93, 94, 41]. Program behavior, however, is not entirely random and often shows significant structure. In [94] and [41], the authors periodically gathered various hardware metrics, like IPC, cache misses, branch misprediction rate with the aim of studying low-level program behavior over time and finding any possible correlation between the metrics. Their findings indicate that, not only does program behavior change, but it also has periods of stable execution interspersed with transitions. During periods of stable execution, the architectural metrics measured are relatively stable. What is more interesting is the fact that the metrics transition in unison, though the nature of the transition might be different (that is the instruction cache miss rate might go up, whereas the IPC might go down). Recognizing the importance of automatically characterizing this behavior in order to exploit it for various purposes (like reducing simulation time, aiding prediction and proactive optimization), various techniques were developed at different levels in the system stack –