VMIL: Workshop on Virtual Machines and Intermediate Languages

Hridesh Rajan[λ], Michael Haupt[ϕ], Christoph Bockisch[β], Robert Dyer[λ]

[λ]Iowa State University, [ϕ]Hasso Plattner Institute, University of Potsdam, and [β]Universiteit Twente

[λ]{hridesh,rdyer}@cs.iastate.edu, [ϕ]michael.haupt@hpi.uni-potsdam.de, and [β]c.m.bockisch@cs.utwente.nl

Abstract

The VMIL workshop is a forum for research in virtual machines (VM) and intermediate languages (IL). It is dedicated to identifying programming mechanisms and constructs currently realized as code transformations or implemented in libraries but should rather be supported at the VM level. Candidates for such mechanisms and constructs include modularity mechanisms (aspects, context-dependent layers), concurrency (threads and locking, actors, software transactional memory), transactions, etc. Topics of interest include the investigation of which such mechanisms are worthwhile candidates for integration with the VM, how said mechanisms can be elegantly (and reusably) expressed at the IL level (e.g., in bytecode), how their implementations can be optimized, and how VM architectures might be shaped to facilitate such implementation efforts.

Categories and Subject Descriptors D.1.5 [*Programming Techniques*]: Object-oriented Programming; D.3.3 [*Programming Languages*]: Language Constructs and Features — Control structures; Procedures, functions, and subroutines; D3.4 [*Language Processors*]: Compilers, Interpreters, Memory Management, Run-Time Environments

General Terms Design, Languages, Performance

Keywords virtual machine, intermediate language, dynamic dispatch, compilation, interpretation, optimization

1. Motivation and Themes

An increasing number of high-level programming language implementations is realized using standard VMs. Recent examples of this trend include the Clojure (Lisp) and Potato (Squeak Smalltalk) projects, which are implemented on top of the Java Virtual Machine (JVM); and also F# (ML) and IronPython, which target the .NET CLR. Making diverse languages—possibly even adopting different paradigms—available on a robust and efficient common platform leverages language interoperability.

Vendors of standard VM implementations have started to adopt extensions supporting this trend from the VM side. For instance, Sun's JVM will include the *invokedynamic* instruction to facilitate a simpler implementation of dynamic programming languages on the JVM.

It has been observed that supporting language constructs in library code, or through code transformations leads to over-generalized results. Thus, efforts are spent to implement the core mechanisms of certain programming paradigms at the VM level, enabling sophisticated optimization by direct access to the running system. This approach has been adopted by several projects aiming at providing support for aspect-oriented programming or dynamic dispatch in general-purpose VMs (Steamloom, Nu, ALIA).

The main themes of this workshop are to investigate which programming language mechanisms are worthwhile candidates for integration with the run-time environment, how said mechanisms can be declaratively (and re-usably) expressed at the intermediate language level (e.g., in bytecode), how their implementations can be optimized, and how VM architectures might be shaped to facilitate such implementation efforts. Possible candidates for investigation include modularity mechanisms (aspects, context-dependent layers), concurrency (threads and locking, actors, software transactional memory), transactions, paradigm-specific abstractions, and combinations of paradigms.

The areas of interest include, but are not limited to, compilation-based and interpreter-based VMs as well as intermediate-language designs with better support for investigated language mechanisms, compilation techniques from high-level languages to enhanced ILs as well as native machine code, optimization strategies for reduction of run-time overhead due to either compilation or interpretation, advanced caching and memory management schemes in support of the mechanisms, and additional VM components required to manage them.

2. Relevance to OOPSLA

To date, many kinds of mechanisms and concepts researched in various communities are mostly reflected in high-level

OOPSLA 2009, October 25–29, 2009, Orlando, Florida, USA.
ACM 978-1-60558-768-4/09/10.

language and library design. The workshop's main goal is the discussion of more natural support for such constructs even within compiled programs, e. g., to improve dynamic optimization, incremental compilation, and debugging. It is expected that language constructs benefit from a more efficient execution and better integration into the development process: both will raise the acceptance of the concepts which in turn activates further research at the conceptual level.

The VMIL workshop has been successfully conducted twice (at the conferences OOPSLA 2008 and AOSD 2007). The scope of the workshop addresses the interests of many communities typically represented at OOPSLA, including the concurrent or transactional programming and the modularization communities. As multicore programming is among the main themes of OOPSLA 2009, we expect the respective community to respond well to this workshop.

3. Invited Talks

Virtual Machine and Intermediate Language Challenges for Parallelism ***by Vivek Sarkar*** A VM specifies the behavior of a system at a high level of abstraction that includes precise semantics for state updates and control flow, but leaves unspecified the low-level software and hardware mechanisms that will be used to implement the semantics. Past VMs have followed the von Neumann execution model by making sequential execution the default at a high level, and supporting parallelism with lower-level mechanisms such as threads and locks. Now that the multicore trend is making parallelism the default execution model for all software, it behooves us as a community to study the fundamental requirements in parallel execution models and explore how they can be supported by higher-level abstractions at the VM level.

In this talk, we discuss five key requirements for parallel VMs: 1) Lightweight asynchronous tasks and communications, 2) Explicit locality, 3) Directed Synchronization with Dynamic Parallelism:, 4) Mutual Exclusion and Isolation with Dynamic Parallelism, and 5) Relaxed Exception semantics for Parallelism. For completeness, these requirements need to be addressed at both the VM and IL levels. We summarize the approach being taken in the Habanero Multicore Software Research project at Rice University (http://habanero.rice.edu) to define a Parallel Intermediate Representation (PIR) to address these requirements, and then identify key research challenges in developing VMs and ILs for parallelism.

Abstraction Without Guilt ***by Steve Blackburn*** While on the one hand systems programmers strive for reliability, security, and maintainability, on the other hand they depend on performance and transparent access to low-level primitives. Abstraction is the key tool for enabling the former but it typically obstructs the latter. This talk addresses this conundrum from three distinct angles; as a producer, a consumer, and an evaluator of high level programming languages, and is based on ten years of experience in each of these roles. I will discuss my experience as a producer, engineering a low-overhead, highly-expressive Java dialect suitable for systems programming; and as a consumer, using Java and object oriented programming principles to build a JVM and memory management subsystem. Key to both of these is the role as an evaluator, measuring and understanding the complex behavior of managed runtime systems. The phrase "abstraction without guilt", coined by Ken Kennedy, nicely captures our philosophy on systems building.

The Maxine Virtual Machine ***by Doug Simon and Ben L. Titzer*** Maxine (http://research.sun.com/projects/maxine/) aims to support VM research and enable fast prototyping of language features and implementation techniques. A meta-circular design implemented in the Java^{TM} programming language blurs the distinction between VM and application and greatly simplifies important VM components. This talk will present details of the VM that make it attractive as a high performance but malleable platform for VM research. We will discuss two of the three compilers in Maxine: C1X, a port of the HotSpot client compiler, and the bootstrap compiler, which is based on continuation passing style. In addition, we will present the Maxine Inspector, a combined object browser and debugger that extensively leverages the meta-circularity of the VM to present a high-fidelity and robust debugging and inspection tool for Maxine.

4. Organization

The organizers of the workshop are as follows. **Hridesh Rajan** is an assistant professor at the Iowa State University. His main research is on AOSD, and on specification and verification languages. **Michael Haupt** is a post-doctoral researcher at the Hasso-Plattner-Institut in Potsdam. His research interests are in improving the modularity of complex systems software. **Christoph Bockisch** is an assistant professor at the University of Twente with a research focus on the design and implementation of programming languages with advanced dispatching mechanisms. **Robert Dyer** is a third year Ph. D. student researching the design of IL models and VM support for advanced modularization techniques.

The program committee of VMIL 2009 consists of **Eric Bodden** (McGill University), **Alex Buckley** (Sun Microsystems), **Andreas Gal** (University of California Irvine), **Doug Lea** (SUNY Oswego), **Stefan Marr** (Vrije Universiteit Brussel), **Filip Pizlo** (Purdue University), **Andreas Sewe** (Technische Universität Darmstadt), **Jan Vitek** (Purdue University), and the organizers.

Acknowledgments

The organization of this workshop was supported in part by the NSF under grants CNS-08-08913, CNS-0627354, CCF-08-46059 and the AOSD-Europe Network of Excellence.