October 15, 201

ReactoR: LLVM-based just-in-time compilation of R programs

Jan Vitek Northeastern University, Computer Science
Tel: 7654092176, j.vitek@neu.edu

Potential sponsor at Google
Karl Millar kmillar@google.com

Abstract

This project will focus on novel hybrid static/dynamic compiler optimizations for a LLVM-based implementation of the R language. These optimizations will be based on speculative-invariants that may be invalidate at runtime. We will produce a theory of speculative analysis and speculative optimizations as well as support for deep unrolling of program transformations.

Research Goals

Scripting languages, like R, are lightweight, dynamic programming languages that maximize productivity by offering high-level abstractions. However, they lose their appeal when requirements stabilize and projects enter their deployment phase. The compromises made to reduce development time make it hard to scale to large data sets or perform computationally intensive tasks. This is certainly the case for R. R is a dynamic language for statistical computing that combines lazy functional features and object-oriented programming. Released in 1995 under a GNU GPL license, R rapidly became the lingua franca for statistical data analysis. Today, there are over 4000 packages available from repositories such as CRAN and Bioconductor. The R-forge web site list 1'102 projects. With its 55 user groups, there are about 2'000 package developers, and over 2 million end users. R's attraction comes from the myriads of open source libraries available on the Internet, and from its dynamic and interactive nature. Unlike previous scientific computing languages which enforced a strong distinction between programming and execution, R allows to manipulate data and experiment with formulas with immediate feedback. Unfortunately, as observed by Ihaka and Lang, R's authors, the application of cutting-edge statistical methodology is limited by the capabilities of the system in which it is implemented. R simply does not scale to the larger problems.

The goals of this project is to improve on the R execution environment and collaborate with Google in the development of a sustainable, next-generation, R environment that will be accepted by the open source R

community and provide increased levels of performance and scalability. **The proposed contribution by the Northeastern team focuses on novel compiler optimizations techniques for a LLVM-based just-in-time compiler for the R language. Our will focus on hybrid static/dynamic analyses that take advantage of information available at runtime to speculatively optimize programs. We will track the flow of speculative fact through the analysis to support powerful optimization backtracking.**

Static program analyses are fundamentally limited by soundness. Analyzing a R program ahead of time can yield very little useful behavioral information about it as too many sequences of actions have to be considered as possible. This is especially so in the presence of reflection and dynamic code generation. This problem has plagued dynamic languages to the point that whole-program analyses for these languages are almost never deployed in production settings.

Our experience is that the pessimism of static analyses can be mitigated with a small amount of runtime information. An example of the use of dynamic techniques to limit reflection can be found in our [OOPSLA12] paper where we show that JavaScript eval statements can be removed with 95% accuracy by a dynamic analysis that observes the first two invocations of eval at any given call site. Then using those two invocation it infers an automaton that will recognize any future invocation of eval from that call site and replace the call to eval with simpler non-reflective calls. Thus with very little runtime information, it becomes possible to eliminate the uncertainty introduced by eval.

The ReactoR project proposes to build on our experience implementing a speculatively optimized version of the R programming language [VEE14]. Our previous work was done within the context of Java virtual machine implementation. For ReactoR we propose to work on top of the LLVM compiler and for better integration with R, start with the CXXR C++ implementation of the R language. We believe this combination of features to be more conducive to adoption by the R community at large.

Within ReactorR we propose to work on combining *static* and *dynamic* program analyses. We will start by working on a small language that we call TinyR to prototype our ideas and then push on to the full R. We propose to start with a model where a *speculative invariant* can be placed at any program point as a result of monitoring of the execution or from domain knowledge obtained from similar program. Speculative invariants are incorporated into program analysis to enable program transformations. Any program transformation step that transitively depends on a speculative invariant is considered to be a *speculative transformation*. During program execution new speculative invariants can be added leading to further optimization, but it is also possible for a speculative invariant to be *invalidated*. Invariant invalidation can cause a cascade of deoptimizations and on-stack replacements. The intellectual challenge of this work is to minimize deoptimization on an invalidation. For this we will need to keep precise track of dependencies and the flow of speculations though the code.

ReactoR will allow developers to write high-level analyses and transformation that would be unsound without it. Examples include sound/best-effort interprocedural static analyses for R in the presence of reflection.

The proposed approach is related to the work done on trace-based compilation, except trace-based compilers usually do not attempt to perform program analysis based on traces. There are similarities with profile driven optimizations with the difference that our approach allows the program to be reanalyzed at any time, and also allows undoing transformations.

Data Policy

All software and data produced within this project will be released in open source format. Results will be be submitted to conferences such as OOPSLA and PLDI.

Budget

A total of 140,000$ where 63,000$ are dedicated to one advanced PhD student for 12 months, 20,000$ to support two undergraduate students for a year (+10840$ for Stipends), 20,000$ + 11656$ Fringe for a month and a half for PI Vitek, 6000$ of travel, 5162$ for a laptop and 3000$ for miscellaneous expenses.

Results from past project

The PI received funding to study security for JavaScript. The students working on this project were Gregor Richards and Fadi Meawad. Fadi joined Google in September 2013. We are happy to report two publications based on that grant. Our OOPSLA12 paper looked at how to reduce the attack surface of JavaScript programs by automatically inferring recognizers for the strings passed into eval, and replacing the reflective call with an equivalent but less powerful operation. The OOPSLA13 paper introduces a new security infrastructure for JavaScript based on the notion of recording execution histories and using these histories for access control check and, possibly, revocation of operations performed by the application. This work is also related to HyDyS — a Google funded project from 2013 — the PI has no results to report; moving from Purdue University to Northeastern University cause the funding to be forfeited.

[OOPSLA12] Meawad, Richards, Morandat, Vitek. *Eval begone! Semi-automated removal of eval from JavaScript programs.* **OOPSLA 2012.**

[OOPSLA13] Richards, Hammer, Jagannathan, Zappa Nardelli, Vitek. *Flexible Access Control Policies with Delimited Histories and Revocation.* **OOPSLA 2013.**

[VEE14] Kalibera, Maj, Morandat, Vitek: *A fast abstract syntax tree interpreter for R*. **VEE 2014.**

Students and postdocs now at Google

Daniel Tang, **BSc**, **MSc**. *Type checking of Safety Critical Java.*

Jeremy Manson, **PostDoc**. *Concurrency and software transactions for Java.*

Hiroshi Yamauchi, **MSc**. *Register allocation and code generation for the Ovm JVM.*

Ben Titzer, **BSc**. *Intermediate representations for Ovm.*

Jason Baker, **MSc**. *The Ovm JVM's bytecode to C ahead of time compiler.*

Nicholas Kidd, **PostDoc**. *Static analysis for real-time concurrent programs*

Fadi Meawad, **PhD**. *Reflection elimination for JavaScript.*