UNIVERSITÉ DE RENNES 1

Direction de la Recherche et de l'Innovation

Appel à projets - Année 2022

Contrats doctoraux «Ordinaires»

ECOLE DOCTORALE : MathSTIC **POLE DE RECHERCHE : Numérique**

Unité de recherche d'accueil

Unité de Recherche : IRISA

Equipe : PACAP

Encadrement

Directeur de thèse : Erven Rohou **Unité de recherche** : IRISA

Employeur/Établissement d'origine : Inria

HDR : ☑ oui ☐ non **Nb de thèses en cours** : 3 (dont 1 soutenance au printemps 2022)

Mail : erven.rohou@inria.fr **Tél**. : 02 99 84 74 93

Co-directeur de thèse : Manuel Serrano **Unité de recherche** : Inria Sophia Antipolis

Employeur/Établissement d'origine : Inria

HDR : ☑ oui ☐ ~~non~~ **Nb de thèses en cours** : 1

Mail : manuel.serrano@inria.fr **Tél.** : 04 92 38 76 39

Sujet

Titre : Profile-Guided optimization for Dynamic Languages

Mots clés : dynamic languages, JavaScript, compilation, microarchitecture, performance

Cotutelle internationale (oui/non) : non **Si oui, préciser l'établissement et le pays de rattachement** :

Descriptif (15-30 lignes) :
Contexte socioéconomique et scientifique ; Les hypothèses et questions posées ; Les grandes étapes de la thèse et démarche ; Approches méthodologiques et techniques envisagées

Context

JavaScript is particularly difficult to implement efficiently because most of its expressions have all sorts of different meanings that involve all sorts of different executions that are not distinguished by any syntactic or type annotation. Checking all the possible interpretations and executing the appropriate one literally, that is treating the language specification as an algorithm, delivers unacceptably slow performance. All fast implementations use alternative strategies. Amongst all the possible interpretations, they favor the one that corresponds to the most frequent situation, for which they elaborate a faster execution plan, and, as importantly, for which they elaborate a fast guard that ensures the preservation of the language semantics. Typically, that is what inline caches and hidden classes achieve [3, 4, 1]. Using a single test, the comparison of the object's hidden class with the inline cache, we know if the property is to be read directly from the object and, if so, at which offset. Until recently it was considered that only dynamic compilers, a.k.a., JIT compilers, can handle dynamic languages efficiently because this heuristic-based strategy requires having the program and the data on hand in order to generate efficient code [5]. Recent studies show more balanced results [8] but whatever the underlying implementation technique used, JIT or AoT (ahead-of-time), the executed code has the same characteristic. The code is cluttered with tests (for the guards), long code (for code duplication and specialization) and memory footprints are large, especially when JIT compilation is used because the dynamic compilation itself is memory demanding. The purpose of this PhD is to study how this impacts the overall performance of JavaScript executions.

Objectives

Modern processors and systems provide users with many sensors and probes the let them finely analyze the performance of their programs at the hardware level [2, 6]. Unfortunately, these tools have so many options and configurations that they are hardly usable by programmers lacking expertise in computer architecture design [7]. The objective of this PhD is to develop methodologies and tools that will help non-expert programmers analyze and understand the performance of their JavaScript programs. All the major industrial implementations, namely Google's V8, Mozilla's SpiderMonkey, and Apple's JavaScriptCore all rely on JIT compilation but they rely on different techniques. Hopc, the ahead-of-time compiler developed at

Inria Sophia, yet again, uses totally different strategies because all its decisions regarding which code to generate and which code to specialize are taken before the execution.
The tools that will be crafted during this PhD will be used to analyze and compare the different JavaScript implementations in order to shed light on aspects of the executions that are generally badly understood. That is, this analysis will compare the behavior of the different executions and their adaptation to actual hardware settings, instead of focusing on raw performance of JavaScript benchmark as most analyses do. More precisely, Hopc has various code generation strategies to translate JavaScript to C. The goal is to compare them with each other, and also against what existing JIT compilers do. For this purpose, the initial idea consists in three steps: first, run the programs and measure wall-clock time; second, run under the control of the perf tool to collect low-level information from hardware counters, such as instruction count, branch misprediction, behavior of memory hierarchy, etc; finally, the statically generated code is run in a micro-architecture simulator, developed at Inria Rennes, capable of pinpointing the behavior of particular instructions, such as branch or memory instructions. As a result of analyzing the data, the objective is to derive recommendation about the best code generation strategy for the static compilation of JavaScript.

[1] R. Artoul. Javascript Hidden Classes and Inline Caching in V8. http://richardartoul.github.io/jekyll/update/2015/04/26/hidden-classes.html, Apr. 2015.
[2] C. Carruth. CppCon 2015, tuning C++: Benchmarks, and CPUs, and compilers! oh my! https://www.youtube.com/watch?v=nXaxk27zwlk, 2015.
[3] C. Chambers and D. Ungar. Customization: Optimizing compiler technology for SELF, a dynamically-typed object-oriented programming language. In Conference Proceedings on Programming Language Design and Implementation, PLDI '89, New York, NY, USA, 1989. ACM.
[4] C. Chambers, D. Ungar, and E. Lee. An efficient implementation of self a dynamically-typed object-oriented language based on prototypes. In 3Conference Proceedings on Object-oriented Programming Systems, Languages and Applications, OOPSLA '89, pages 49–70, USA, 1989. ACM.
[5] G. Dot et al. Removing checks in dynamically typed languages through efficient profiling. In Proceedings of the 2017 International Symposium on Code Generation and Optimization, CGO '17, pages 257–268. IEEE Press, 2017.
[6] B. Gregg. The flame graph. Commun. ACM, 59(6):48–57, May 2016.
[7] D. Patterson and J. Hennessy. Computer Architecture, a Quantitative Approach. Morgan Kaufmann, 2nd edition, 1996.
[8] M. Serrano. Of javascript AOT compilation performance. Proceedings of the ACM on Programming Languages, Aug. 2021.