LLVM and JitBuilder: Measuring and Comparing the Overhead of two API-based JIT Frameworks

Eric Coffin
Faculty of Computer Science
University of New Brunswick
Fredericton, NB, Canada
eric.coffin@unb.ca

ABSTRACT

Just-in-Time compilation has allowed for significant performance gains during the run-time of applications. Two popular open-source JIT frameworks are LLVM MCJIT and OMR JitBuilder, both of which can be embedded within applications, offering interaces to define and generate native code at run-time. LLVM is a collection of modular compiler and toolchain components, while MCJIT is a framework based on these components to provide JIT compilation. Similarly, JitBuilder is a JIT framework based on the OMR compiler and runtime components. In this report we discuss the different approaches these two frameworks employ. In addition, we measure the overhead of each framework while compiling and then executing a small handful of functions. We found that while LLVM required a larger memory footprint, in certain cases it was able to generate code more quickly. Furthermore, the code LLVM generated typically offered high throughput. JitBuilder, a relatively young project compared to LLVM, does not currently expose the underlying configuration of TRJIT. Instead, it locks the compilation level to the *warm* setting, thus limiting the oppourtunities for optimizations.

CCS CONCEPTS

• **Software and its engineering → Compilers**.

KEYWORDS

compilers, just-in-time, optimization

1 INTRODUCTION

Just-in-time compilation, or JIT compilation, is a technique to improve the run-time binary translation of an application. Using information collected from the running application, JIT compilers can further optimize generated code. For instance, by collecting profile information on executed code paths, JIT compilers can generate code that is optimized for hot-paths. A JIT compiler might also inline entire blocks of code, or generate an inline cache to speed up the dispatch of polymorphic method calls. Considering JIT compilers may also perform many of the optimizations found in a static compiler such as common sub-expression elimination, loop unrolling and constant propagation, to name a few, an application developer, or language designer interested in enhancing run-time performance by adding a JIT compiler, will have their work cut out for them. In addition, for projects targeting multiple architectures, considering JIT compilers generate native, or architecture specific code, the effort required will increase significantly. It is no surprise then, that libraries or frameworks, encapsulating proven, high-performance JIT compilers are now commonly available to application developers.

In this report, we will look at two such JIT frameworks: LLVM MCJIT and OMR JitBuilder.

2 BACKGROUND

2.1 LLVM

LLVM.

2.2 JITBuilder

JITBuilder.

3 METHODOLOGY

Methodology

3.1 Results

4 RELATED WORK

Related work.

5 FUTURE WORK

Future Work

6 SUMMARY

Summary

7 ACKNOWLEDGEMENTS

This research was conducted within the Centre for Advanced Studies—Atlantic, Faculty of Computer Science, University of New Brunswick. The authors are grateful for the colleagues and facilities of CAS Atlantic in supporting our research. The authors would like to acknowledge the funding support provided by the Atlantic Canada Opportunities Agency (ACOA) through the Atlantic Innovation Fund (AIF) program. Furthermore, we would also like to thank the New Brunswick Innovation Foundation for contributing to this project.

Conference'17, July 2017, Washington, DC, USA

REFERENCES