Eclipse OMR Compiler Architecture

February 28, 2018

Moderator: Daryl Maier (: 0xdaryl)

https://github.com/eclipse/omr/issues/2316

Agenda

- https://github.com/eclipse/omr/issues/2325

- Meeting introduction (@0xdaryl)
- OMR compiler architecture community focus areas (@0xdaryl)
- Problems transmuting TR::Nodes discussion (#2338) (@fjeremic)

Welcome!

- Purpose of this meeting
  - Regular community interlock for those interested in contributing, advancing, and learning about Eclipse OMR compiler technology
  - Facilitate discussion on issues and pull requests
  - Introduce and discuss new technology prior to contribution
- Format
  - Open discussion
  - Agenda topics only
- Scope of topics
  - Limited to technical topics pertaining to OMR compiler technology

OMR Compiler Architecture Community Focus Areas

JitBuilder

Problem Statement

Creating a new JIT compiler or native code generator from scratch is time consuming and complicated. JitBuilder makes it easier to write a new JIT compiler for a language runtime or native code generator primarily by simplifying how operations are described to the compiler. The library will also evolve over time to create new structures to help translate the design of a Virtual Machine to a JIT compiler. Eventually, we aim to use a single description of a Virtual Machine to drive the creation of both efficient interpreters and JIT compilers.

JitBuilder

High-Level Solution

- Separation of the Client API from the implementation
  - Introduce semantic versioning to facilitate Eclipse OMR release(s)
- Different language bindings: developing C, C++, Java, …
- Writers and readers for JitBuilder IL (JBIL)
- Platform enablement: Linux x86, macOS already working
- Improve AOT compilation story
- Develop JitBuilder APIs for profiling, expression of ”class” hierarchy/relationships, guards and runtime assumptions, OSR, inlining, interaction with GC, as these pieces migrate into OMR

Increase enablement of language-agnostic compiler technology

Problem Statement

One of the value propositions of Eclipse OMR is to leverage years of investment in OpenJ9 Java technology to benefit other language runtimes. While many features from OpenJ9 have surfaced in the OMR project there are still key technology pieces that deliver significant performance and functional value to Java that have yet to be contributed. The incubation ports of OMR technology to runtimes such as Lua and Swift have already discovered that technology such as value profiling, runtime assumptions, on-stack replacement, and inlining are essential for significant performance gains but these features are not yet generally consumable by non-Java runtimes in Eclipse OMR.

Increase enablement of language-agnostic compiler technology

High-Level Solution

- Generalize runtime assumptions framework, recompilation, code patching, and class hierarchy representation from OpenJ9
- Generalize on-stack replacement technology for non-Java
- Generalize Ahead-of-Time (AOT) technology
- Build out a re-usable and configurable inlining optimization free from Java heuristics

Improve consumability of Eclipse OMR compiler technology

Problem Statement

The technology available in Eclipse OMR is powerful, but its complexity provides a steep learning curve to those that wish to leverage it in a language runtime. OpenJ9 provides the best working example of how to consume the technology, but it also overwhelms in its complexity. Clear architectural guidance on what each component is responsible for and how they work together, documented interfaces, and working examples consuming the technology are essential to increase adoption of Eclipse OMR technology.

Improve consumability of Eclipse OMR compiler technology

High-Level Solution

- Develop a specification for Testarossa IL, type system, and aliasing
  - Enhance validators and unit test cases to ensure correctness
- Improve integration of Eclipse OMR technology components (e.g., compiler, GC, JitBuilder, port, etc.) to provide a consistent means of consumption
  - https://github.com/eclipse/omr/issues/1642
- Improve conceptual integrity of key technology components (e.g., compiler, JitBuilder, etc.) and subcomponents (e.g., code generators, code cache, optimizations, etc.) through documented interfaces and restructuring of the code to abide by those interfaces
- Continue developing Base9 (https://github.com/b9org/b9) as the primary educational platform for Eclipse OMR technology. Develop in-depth tutorials, examples, and documentation of varying complexity to demonstrate the use of compiler technology in runtime environments

Prepare compiler technology for new targets and compilation environments

Problem Statement

Part of the OMR compiler technology's value proposition is to target as many processor architectures as possible. In addition, the evolution of compiler technology in cloud environments is toward a microservices model. Compiler technology needs to be adaptable to provide compilation services out of process from its language runtime, and provide the ability to target multiple processors from the same compilation host. While the Testarossa compiler technology did support cross compilation for the same language target in its early embedded days, the key infrastructural guarantees to support generating code for different processor architectures have long eroded and must be restored for a proper cross-compilation solution. Also, the interface between the compiler and language runtime must be better architected and made more consistent.

Prepare compiler technology for new targets and compilation environments

High-Level Solution

- Provide an AArch64 compiler backend to complete support for ARM
  - Part of a broader effort to add AArch64 support to OMR
  - https://github.com/eclipse/omr/issues/2327
  - Multi-party effort: University of New Brunswick, IBM Tokyo & IBM Canada
- Complete refactoring of legacy compiler FrontEnd interface
  - Support the notion of different compilation environments (e.g., native compilation, cross compilation, AOT compilation, JITaaS compilation)
- Complete cross compilation infrastructural tasks described by the epic https://github.com/eclipse/omr/issues/1674

Improve testability of compiler technology

Problem Statement

Development of unit tests to exercise features in the compiler technology has long been a difficult task. The traditional technique has been to either rely on a large corpus of tests hoping that eventually the compiler code you're interested in testing will eventually be executed, or to develop a high-level test case structured in such a way that (when coupled with command-line options) baits the compiler into taking the path you wish to test. Such tests are often fragile and time-consuming to develop which discourages developers from creating tests. The development of Tril allows much more focused testing of compiler technology with a low development cost. However, the technology is still immature and needs more features to become the de facto solution for compiler unit test case development.

Improve testability of compiler technology

High-Level Solution

- Expand Tril's scope to include all of Eclipse OMR's IL and datatypes
- Replace functional verification compiler tests with Tril equivalents
- Architect a means to control testing behaviour based on capabilities of the target processor or environment
- Document techniques and best practices for contributors to create unit tests to supplement their code when appropriate