Sulong: Fast LLVM IR Execution on the JVM with Truffle and Graal

Manuel Rigger @RiggerManuel
PhD student at Johannes Kepler University Linz, Austria
Why Do We Need A(nother) LLVM IR Interpreter?

Compile-time

Link-time

Run-time

Offline

Speculative optimizations?

Motivation Example: Function Pointer Calls

```c
void bubble_sort(int *numbers, int count, (*compare)(int a, int b)) {
    for (int i = 0; i < count; i++) {
        for (int j = 0; j < count - 1; j++) {
            if (compare(numbers[j], numbers[j+1]) > 0) {
                swap(&numbers[j], &numbers[j+1]);
            }
        }
    }
}

int ascending(int a, int b) { return a - b; }

int descending(int a, int b) { return b - a; }
```

Sulong

• LLVM IR interpreter running on the JVM
  • With dynamic optimizations and JIT compilation!

• Available under a BSD 3-Clause License
  • https://github.com/graalvm/sulong
  • Contributions are welcome!

• Sulong: Chinese for velocisaurus
  • 速: fast, rapid
  • 龙: dragon
Truffle Multi-Language Environment

Java
JavaScript
Ruby
R
C
Scala

LLVM

http://www.github.com/graalvm

[1]
define i32 @ascending(i32 %a, i32 %b) {
    %1 = sub nsw i32 %a, %b
    ret i32 %1
}
Truffle and Graal

Node Rewriting for Profiling Feedback

Node Transitions
- Uninitialized
- Integer
- String
- Double
- Generic

Compilation using Partial Evaluation

AST Interpreter
Uninitialized Nodes

Rewritten Nodes

Compiled Code
Truffle and Graal

Deoptimization to AST Interpreter

Node Rewriting to Update Profiling Feedback

Recompilation using Partial Evaluation
Example 1: Value Profiling

```
expectedValue = memory[ptr];
deoptimizeAndRewrite();
```

```
currentValue = memory[ptr];
if (currentValue == expectedValue) {
    return expectedValue;
} else {
    deoptimizeAndRewrite();
}
```

```
return memory[ptr];
```
Example 2: Polymorphic Function Pointer

Inline Caches

\[
\text{compare}(a, b) > 0
\]

No call 1 call 2 calls >2 calls

Uninitialized CallNode Direct CallNode Direct CallNode Indirect CallNode

\[
\text{if (compare == &ascending) }
\]

\[
\text{return ascending}(a, b);
\]

\[
\text{else if (compare == &descending) }
\]

\[
\text{return descending}(a, b);
\]

\[
\text{else }
\]

\[
\text{deoptimizeAndRewrite();}
\]
Function Pointer Call Inlining

bubble_sort

descending

ascending

bubble_sort

descending  ascending
Demo
Getting started

• Download the mx build tool

```bash
$ hg clone https://bitbucket.org/allr/mx
$ export PATH=$PWD/mx:$PATH
```

• Clone the repo and build the project

```bash
$ git clone https://github.com/graalvm/sulong
$ cd sulong
$ mx build
```

• Compile and run a program

```bash
$ mx su-clang -S -emit-llvm -o test.ll test.c
$ mx su-run test.ll
```
Developing with mx

• Generate Eclipse project files (also available for other IDEs)
  
  $ mx eclipseinit

• Quality tools
  
  $ mx checkstyle/findbugs/pylint/...

• run Sulong tests
  
  $ mx su-tests

• Eclipse remote debugging (port 5005)
  
  $ mx su-debug test.ll
Compilation

• Textual information about which LLVM functions are compiled

   $ mx su-run test.ll -Dgraal.TraceTruffleCompilation=true

• View Truffle and Graal graphs

   $ mx igv
   $ mx su-run test.ll -Dgraal.Dump=Truffle
Example: Truffle Graph
Implementation of Memory

• Unmanaged mode
  • Heap allocation: by native standard libraries
  • Stack allocation: Java Unsafe API

• Graal Native Function Interface for library interoperability
Current State

• Performance: room for improvement on most benchmarks
• Completeness: mostly focused on C so far
  • Missing: longjmp/setjmp, inline assembly, full support of 80 bit floats
  • Can execute most of the gcc.c-torture/execute benchmarks
Outlook

• Low overhead security-related instrumentations
  → Graal is specialized to perform optimizations for operations like bounds or type checks
  • Memory safety via allocating on the Java heap
  • Tracking of integer overflows

• Full Truffle integration
  • Debugger with source code highlighting
  • Language interoperability
Q/A @RiggerManuel

• Thanks for listening!
Attributions

• [1] The JRuby logo is copyright (c) Tony Price 2011, licensed under the terms of Creative Commons Attribution-NoDerivs 3.0 Unported (CC BY-ND 3.0)