Graal and Truffle: Modularity and Separation of Concerns as Cornerstones for Building a Multipurpose Runtime

Thomas Wuerthinger
Oracle Labs
@thomaswue

24-April-2014, Keynote at MODULARITY in Lugano
Disclaimer

The following is intended to provide some insight into a line of research in Oracle Labs. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described in connection with any Oracle product or service remains at the sole discretion of Oracle. Any views expressed in this presentation are my own and do not necessarily reflect the views of Oracle.
Agenda

- Graal
- Truffle
- Community
- Q&A
Dimensions of Extensibility

Architectures
- X86, SPARC, HSAIL, PTX, ...

Host Runtimes
- HotSpotVM, SubstrateVM, ...

Compilation Policy
- Baseline, Mid-tier, Optimizing, Vectorizing, ...

Programming Languages
- Java, JavaScript, Python, R, Ruby, C, Smalltalk, ...

Graal
Modularity

Fine-grained modular structure with Rich Client Platforms like Eclipse or NetBeans as role models.
Specific to Host Runtime

- Field/Array Access
  - object/array layout, read/write barriers, ...

- Allocation
  - garbage collector, thread-local buffer, ...

- Type Checks
  - class hierarchy organization, ...

- Locking
  - monitor system, monitor enter/exit, ...

- JDK intrinsics
  - hashCode, clone, reflection, ...

- Invocations

- Safepoints
Levels of Lowering

Java
Lowered
Runtime-Specific
ISA-Specific

JavaArrayStore
NullCheck
Length
BoundsCheck
TypeCheck
Store
NullCheck
Read
UCmp
Read
Cmp
Barrier
Write
Read&Check
UCmp
Read&Cmp
Shift
Write
Write
Snippets for Graph Construction

Manual construction:

```java
Node max(ValueNode a, ValueNode b) {
    IfNode ifNode = new IfNode(new IntegerLessThanNode(a, b));
    ifNode.trueSuccessor().setNext(new ReturnNode(a));
    ifNode.falseSuccessor().setNext(new ReturnNode(b));
    return ifNode;
}
```

Expression as snippet:

```java
int max(int a, int b) {
    if (a > b) return a;
    else return b;
}
```
Simple API

- Can capture statically
- Limited flexibility
Callback API

API User

request including callback

callback with data

callback with data

callback with data

request returns

API Provider

✓ High flexibility

× Cannot capture statically
Snippet API

API User

API Provider

request

code as data

✓ High flexibility  ✓ Can capture statically
Snippet Lifecycle

**Preparation**
- Bytecodes: `aload_0 getfield ifne 10 aload_1 arraylength ...
- Frequency: Once
- Prepared IR Graph

**Specialization**
- Frequency: Few Times
- Specialized IR Graphs

**Instantiation**
- Frequency: Many Times
- ...
Snippet Example: Convert

@Snippet
static int f2i(float input, int result) {
    if (probability(SLOW_PATH,
                    result == Integer.MIN_VALUE)) { 
        if (Float.isNaN(input)) {
            return 0;
        } else if (input > 0.0f) {
            return Integer.MAX_VALUE;
        }
    }
    return result;
}
Agenda

- Graal
- **Truffle**
- Community
- Q&A
Technical Approach
Speculate and Optimize…

Node Rewriting for Profiling Feedback

Node Transitions

AST Interpreter Uninitialized Nodes

Compilation using Partial Evaluation

Compiled Code

AST Interpreter Rewritten Nodes

Uninitialized Integer

Uninitialized Double

Generic

String

Integer

Double

Generic

Uninitialized
Technical Approach

… and Deoptimize and Reoptimize!

Deoptimization to AST Interpreter

Node Rewriting to Update Profiling Feedback

Recompilation using Partial Evaluation
Technical Approach
Three main parts for driving partial evaluation

- Limit partial evaluation expansion
  - Annotation @SlowPath on a method stops the inclusion of a method in the expansion.

- Dynamic speculation
  - Call to CompilerDirectives.transferToInterpreter() advises the partial evaluator to stop and place a deoptimization exit.

- Global speculation
  - Assumption objects can be used for global speculations about the system state. Checking the assumption in compiled code poses no runtime overhead.
Peak Performance: JavaScript

Speedup relative to V8

Selection of benchmarks from Google's Octane benchmark suite v1.0
latest versions of V8, Truffle, and SpiderMonkey as of December 2013
Peak Performance: C

Speedup relative to GCC O0

Grimmer, Rigger, Schatz, Stadler, Mössenböck: 
TruffleC: Dynamic Execution of C on the Java Virtual Machine; to be submitted
Agenda

- Graal
- Truffle
- Community
- Q&A
Graal OpenJDK Project

http://openjdk.java.net/projects/graal/

- Development of Graal/Truffle core artifacts and APIs
- Highly active: 30+ contributors over last 12 months
- Highly modular: 80+ individual modules
## Research Areas

<table>
<thead>
<tr>
<th>Language Implementation</th>
<th>Experimentation with new language features, new languages, new execution models</th>
<th>Truffle Interpreters</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>General Language Research</strong></td>
<td>Language-independent instrumentation, cross-language research, automatic partial evaluation experiments</td>
<td>Truffle</td>
</tr>
<tr>
<td><strong>Compiler Construction</strong></td>
<td>Core compiler construction research, heterogenuous computing, advanced architectures and backends</td>
<td>Graal</td>
</tr>
</tbody>
</table>
Graal/Truffle Related Research Projects (1)

- **TruffleRuby**
  - Development in the JRuby repository (lead Chris Seaton).
  - [https://github.com/jruby/jruby](https://github.com/jruby/jruby)
  - [http://blog.jruby.org/2014/01/truffle_graal_high_performance_backend/](http://blog.jruby.org/2014/01/truffle_graal_high_performance_backend/)

- **FastR**
  - Joint effort of a group from Purdue University (Prof. Jan Vitek) and a team at Oracle Labs (lead Michael Haupt).
  - [https://bitbucket.org/allr/fastr](https://bitbucket.org/allr/fastr)

- **ZipPy**
  - Development by a group from University of California, Irvine (Prof. Michael Franz).
  - [https://bitbucket.org/sslab/zippy](https://bitbucket.org/sslab/zippy)

- **TruffleSOM**
  - Development by Stefan Marr at: [https://github.com/smarr/](https://github.com/smarr/)
Graal/Truffle Related Research Projects (2)

- C and Language Interoperability
  - Experiment by students at JKU Linz (Matthias Grimmer and Manuel Rigger).

- JavaScript
  - Effort done by the core Graal/Truffle team.

- Debugging
  - Effort by Micheal van de Vanter from Oracle Labs.

- SubstrateVM
  - Team at Oracle Labs led by Christian Wimmer is developing an alternative host runtime.

- Graal IR Instrumentation
  - Research by Yudi Zheng (USI Lugano) on instrumenting Graal IR.

- GPU Offload
  - Research by Christopher Dubach et al. from the University of Edinburgh.
  - Graal is the compiler of choice for Project Sumatra (HSAIL/PTX offload).
Your Language or Compiler Extension?

http://openjdk.java.net/projects/graal/

graal-dev@openjdk.java.net

$ hg clone http://hg.openjdk.java.net/graal/graal
$ cd graal
$ ./mx --vm server build
$ ./mx ideinit
$ ./mx --vm server unittest SumTest

- Graal Resources
  https://wiki.openjdk.java.net/display/Graal/Main

- Truffle API License: GPLv2 with Classpath Exception
- Graal License: GPLv2
Acknowledgements

**Oracle Labs**
Danilo Ansaloni
Daniele Bonetta
Laurent Daynès
Erik Eckstein
Michael Haupt
Peter Kessler
David Leibs
Mick Jordan
Tom Rodriguez
Roland Schatz
Chris Seaton
Doug Simon
Lukas Stadler
Michael Van De Vanter
Adam Welc
Christian Wimmer
Christian Wirth
Mario Wolczko
Thomas Würthinger
Laura Hill

**Interns**
Miguel Garcia Gutierrez
Shams Imam

**Oracle Labs**
Stephen Kell
Gregor Richards
Rifat Shariyar

**JKU Linz**
Prof. Hanspeter Mössenböck
Stefan Anzinger
Gilles Duboscq
Josef Eisl
Matthias Grimmer
Christian Häubl
Josef Haider
Christian Humer
Christian Huber
David Leopoldseder
Manuel Rigger
Georg Schmid
Bernhard Urban
Andreas Wöß

**LaBRI**
Floréal Morandat

**University of California, Irvine**
Prof. Michael Franz
Codrut Stancu
Gulfem Savrun Yeniceri
Wei Zhang

**Purdue University**
Prof. Jan Vitek
Tomas Kalibera
Petr Maj
Lei Zhao

**T. U. Dortmund**
Prof. Peter Marwedel
Helena Kotthaus
Ingo Korb

**University of California, Davis**
Prof. Duncan Temple Lang
Nicholas Ulle

**University of Edinburgh**
Christophe Dubach
Juan José Fumero Alfonso
Ranjeet Singh
Toomas Remmelg

**And many more...**
http://openjdk.java.net/projects/graal/

graal-dev@openjdk.java.net

@thomaswue