Node.js
Just as fast, higher, stronger with
GraalVM™

Oleg Šelajev
Oracle Labs
@shelajev
Safe Harbor Statement

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.
Automatic transformation of interpreters to compilers

GraalVM™

Embeddable in native or managed applications

Languages: Kotlin, Scala, Java™, JS, Ruby, R, Python™, C++
Community Edition (CE)

GraalVM CE is available for free for development and production use. It is built from the GraalVM sources available on GitHub. We provide pre-built binaries for GraalVM CE for Linux on x86 64-bit systems.

DOWNLOAD FROM GITHUB

Enterprise Edition (EE)

GraalVM EE provides additional performance, security, and scalability relevant for running critical applications in production. It is free for evaluation uses and available for download from the Oracle Technology Network. We provide binaries for GraalVM EE for Linux or Mac OS X on x86 64-bit systems.

DOWNLOAD FROM OTN
Launched earlier this month: GraalVM Enterprise 19.0

- More performance
- Smaller footprint
- Managed runtime for better isolation when running native code
- Oracle Enterprise Support 7x24x365
Why GraalVM?

Fast Java, Scala, Kotlin, Groovy, Clojure...

Instant startup, low footprint

Polyglot & embeddable VM

Interoperability between languages: LLVM, Python, Ruby, R
Why GraalVM? version

Let's find out
<table>
<thead>
<tr>
<th>Command</th>
<th>Command</th>
<th>Command</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>javafxpackager</td>
<td>jstack</td>
<td>rmic</td>
</tr>
<tr>
<td>Rscript</td>
<td>javah</td>
<td>jstat</td>
<td>rmid</td>
</tr>
<tr>
<td>appletviewer</td>
<td>javap</td>
<td>jstatd</td>
<td>rmiregistry</td>
</tr>
<tr>
<td>clang-sandboxed</td>
<td>javapackager</td>
<td>jvisualvm</td>
<td>ruby</td>
</tr>
<tr>
<td>extcheck</td>
<td>jcmd</td>
<td>keytool</td>
<td>schemagen</td>
</tr>
<tr>
<td>gem</td>
<td>jconsole</td>
<td>lli</td>
<td>serialver</td>
</tr>
<tr>
<td>gemasrv</td>
<td>jdb</td>
<td>native-image</td>
<td>servertool</td>
</tr>
<tr>
<td>graalpython</td>
<td>jdeps</td>
<td>native2ascii</td>
<td>testrb</td>
</tr>
<tr>
<td>gu</td>
<td>jhat</td>
<td>node</td>
<td>tnameserv</td>
</tr>
<tr>
<td>idealgraphvisualizer</td>
<td>jinfo</td>
<td>npm</td>
<td>truffleruby</td>
</tr>
<tr>
<td>idlj</td>
<td>jjs</td>
<td>orbd</td>
<td>unpack200</td>
</tr>
<tr>
<td>irb</td>
<td>jmap</td>
<td>pack200</td>
<td>wsgen</td>
</tr>
<tr>
<td>jar</td>
<td>jmc</td>
<td>policytool</td>
<td>wssimport</td>
</tr>
<tr>
<td>jarsigner</td>
<td>jps</td>
<td>polyglot</td>
<td>xjc</td>
</tr>
<tr>
<td>java</td>
<td>jrunscript</td>
<td>rake</td>
<td></td>
</tr>
<tr>
<td>javac</td>
<td>js</td>
<td>rdoc</td>
<td></td>
</tr>
<tr>
<td>javadoc</td>
<td>jsadepbugd</td>
<td>ri</td>
<td></td>
</tr>
</tbody>
</table>
Without the JVM

GraalVM JIT Compiler

Truffle Framework

Sulong (LLVM)
V8 Dev Blog — https://v8.dev/blog/ignition-interpreter
V8 Dev Blog — [https://v8.dev/blog/ignition-interpreter](https://v8.dev/blog/ignition-interpreter)
function TreeNode(execute, children) {
    this.execute = execute;
    this.children = children;
}
worklist.push(rootMethod.start)
do {
    node = worklist.peek();

    if (node.hasNotEvaluatedSuccessors()) {
        worklist.pushAll(node.successors)
    } else {
        worklist.pop();
        node.evaluate();
    }
} while (worklist.notEmpty)
Execution
Partial Evaluation of Computation Process—
An Approach to a Compiler-Compiler

YOSHIHIKO FUTAMURA
Central Research Laboratory, Hitachi, Ltd., Kokubunji, Tokyo, Japan 185

Abstract. This paper reports the relationship between formal description of semantics (i.e., interpreter) of a programming language and an actual compiler. The paper also describes a method to automatically generate an actual compiler from a formal description which is, in some sense, the partial evaluation of a computation process. The compiler-compiler inspired by this method differs from conventional ones in that the compiler-compiler based on our method can describe an evaluation procedure (interpreter) in defining the semantics of a programming language, while the conventional one describes a translation process.
function pow (x, y) {
    return x ** y;
}

function pow (x, 2) {
    return x * x;
}
Executable
Speculation and Deoptimization

- Collect profiling feedback
- Optimize using partial evaluation assuming stable profiling feedback
- Deoptimize if profiling feedback is invalid and reprofile
Stability

The diagram above illustrates the stability of functions with stable specializations across different programming languages. The x-axis represents the number of function invocations, while the y-axis shows the percentage of functions with stable specializations.

- **JavaScript** (blue line with 'x' markers)
- **Ruby** (red line with square markers)
- **R** (green line with triangle markers)

The graph shows that as the number of function invocations increases, the percentage of functions with stable specializations also increases for all languages. However, JavaScript and Ruby tend to reach their saturation point faster compared to R.

This indicates that, in general, functions with stable specializations require a certain number of invocations to stabilize, and languages like JavaScript and Ruby may be more efficient in this regard.
Warmed-up ClojureScript unit tests

https://github.com/graalvm/graaljs/issues/29
## ECMAScript compatibility

<table>
<thead>
<tr>
<th>Feature name</th>
<th>Edge 1.7</th>
<th>FF.61</th>
<th>CH.67</th>
<th>Node ≥8.10</th>
<th>JJS 1.8</th>
<th>JJS 1.10</th>
<th>GraalVM 1.0-2</th>
<th>96%</th>
<th>98%</th>
<th>98%</th>
<th>97%</th>
<th>7%</th>
<th>28%</th>
<th>97%</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>default function parameters</td>
<td>7/7</td>
<td>7/7</td>
<td>7/7</td>
<td>7/7</td>
<td>0/7</td>
<td>4/7</td>
<td>7/7</td>
<td>85%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rest parameters</td>
<td>5/5</td>
<td>5/5</td>
<td>5/5</td>
<td>5/5</td>
<td>0/5</td>
<td>0/5</td>
<td>5/5</td>
<td>84%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>spread (...</td>
<td>operator</td>
<td>15/15</td>
<td>15/15</td>
<td>15/15</td>
<td>15/15</td>
<td>0/15</td>
<td>0/15</td>
<td>83%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>object literal extensions</td>
<td>6/6</td>
<td>6/6</td>
<td>6/6</td>
<td>6/6</td>
<td>0/6</td>
<td>2/6</td>
<td>6/6</td>
<td>82%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>for of loops</td>
<td>9/9</td>
<td>9/9</td>
<td>9/9</td>
<td>9/9</td>
<td>0/9</td>
<td>4/9</td>
<td>9/9</td>
<td>81%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>octal and binary literals</td>
<td>4/4</td>
<td>4/4</td>
<td>4/4</td>
<td>4/4</td>
<td>0/4</td>
<td>2/4</td>
<td>4/4</td>
<td>80%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>template literals</td>
<td>5/5</td>
<td>5/5</td>
<td>5/5</td>
<td>5/5</td>
<td>0/5</td>
<td>3/5</td>
<td>5/5</td>
<td>79%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RegExp &quot;y&quot; and &quot;u&quot; flags</td>
<td>5/5</td>
<td>5/5</td>
<td>5/5</td>
<td>5/5</td>
<td>0/5</td>
<td>0/5</td>
<td>5/5</td>
<td>78%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>destructuring declarations</td>
<td>22/22</td>
<td>22/22</td>
<td>22/22</td>
<td>22/22</td>
<td>0/22</td>
<td>0/22</td>
<td>22/22</td>
<td>77%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>destructuring assignment</td>
<td>24/24</td>
<td>24/24</td>
<td>24/24</td>
<td>24/24</td>
<td>0/24</td>
<td>0/24</td>
<td>24/24</td>
<td>76%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>destructuring parameters</td>
<td>23/24</td>
<td>24/24</td>
<td>24/24</td>
<td>24/24</td>
<td>0/24</td>
<td>0/24</td>
<td>24/24</td>
<td>75%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unicode code point escapes</td>
<td>2/2</td>
<td>2/2</td>
<td>2/2</td>
<td>2/2</td>
<td>0/2</td>
<td>0/2</td>
<td>2/2</td>
<td>74%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>new.target</td>
<td>2/2</td>
<td>2/2</td>
<td>2/2</td>
<td>2/2</td>
<td>0/2</td>
<td>0/2</td>
<td>2/2</td>
<td>73%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Interoperability Example

```
a = obj.prop
```

"obj" is not a JavaScript object

Message Resolution

Uses an inline cache entry for every language.
boolean isNull(Object receiver);
boolean isBoolean(Object receiver);
boolean asBoolean(Object receiver);
boolean isExecutable(Object receiver);
Object execute(Object receiver, Object... arguments);
boolean isInstantiable(Object receiver);
Object instantiate(Object receiver, Object... arguments);
boolean isString(Object receiver);
String asString(Object receiver);
boolean isNumber(Object receiver);
boolean fitsInByte(Object receiver);
boolean fitsInShort(Object receiver);
boolean fitsInInt(Object receiver);
boolean fitsInLong(Object receiver);
boolean fitsInFloat(Object receiver);
boolean fitsInDouble(Object receiver);
byte asByte(Object receiver);
short asShort(Object receiver);
int asInt(Object receiver);
long asLong(Object receiver);
float asFloat(Object receiver);
boolean isObject(Object receiver);
Object getMembers(Object receiver, boolean includeInternal);
Object getMembers(Object receiver);
boolean isMemberReadable(Object receiver, String member);
Object readMember(Object receiver, String member);
boolean isMemberModifiable(Object receiver, String member);
void writeMember(Object receiver, String member, Object value);
boolean isMemberRemovable(Object receiver, String member);
void removeMember(Object receiver, String member);
boolean isMemberInvokable(Object receiver, String member);
Object invokeMember(Object receiver, String member, Object... arguments);
boolean isMemberInternal(Object receiver, String member);
boolean isMemberWritable(Object receiver, String member);
boolean isMemberExisting(Object receiver, String member);
Object isArray(Object receiver);
Object readElement(Object receiver, long index);
long getArraySize(Object receiver);
boolean isElementReadable(Object receiver, long index);
void writeElement(Object receiver, long index, Object value);
void removeElement(Object receiver, long index);
boolean isElementModifiable(Object receiver, long index);
boolean isElementInsertable(Object receiver, long index);
boolean isElementRemovable(Object receiver, long index);
boolean isElementExisting(Object receiver, long index);
boolean isPointer(Object receiver);
long asPointer(Object receiver);
Object toNative(Object receiver);
const Thread = Java.type('java.lang.Thread');
const t = new Thread(function run() {
    console.log('hello from another thread!');
})

    t.start();
    t.join();
const Thread = Java.type('java.lang.Thread')
const t = new Thread(function run()
  console.log('hello from another thread!')

);
t.start();
t.join();
• An arbitrary number of JS runtimes can be used by one thread at a time.
• Concurrent access to Java objects is allowed
• Concurrent access to JavaScript objects is not allowed

let w = new Worker(`
    const JavaClass = Java.type('my.very.important.JavaClass');
    const { parentPort } = require('worker_threads');
    parentPort.postMessage(JavaClass.someVeryLongCall());
    
    `, {eval:true});

w.on('message', (m) => {
    console.log('Got data from Java, via worker thread:' + m);
});
Composing Tools

- Debugger
- Profiler
- Coverage
- IDE integration

Languages:
- C
- C++
- Python
- Java
- R
- Scala
- JavaScript
- Ruby
AST Node (vertex) Tagging Example

```javascript
function foo(a, b) {
    a = b + 42
}
```

```java
interface InstrumentableNode {
    boolean isInstrumentable();
    boolean hasTag(Class<Tag> tag);
    WrapperNode createWrapper(ProbeNode probe);
    ...
}
```
Conditional Breakpoint Example

- **Asynchronous breakpoint Installation**
  - installs wrapper and triggers external invalidation of the AST

- **Next Execution**
  - lazily installs breakpoint and inserts condition AST

- **On Enter**
  - Probe
module FunctionValidator

def self.validate(expr)
  expr.split(/\+\|\-\|\*\|/).each do |term|
    term = "" if term == '*'
    is_number = Float(term) rescue false
    is_number = false
    unless is_number
      unless ALLOWED_FUNCTIONS.include?(term)
        return "Unknown expression: #{term}"
      end
    end
  end
  return nil
end

Polyglot.export :Validator, FunctionValidator

{ Line 8, Column 1 }
Why GraalVM?

Polyglot applications: Java, Python, Ruby, R, LLVM

JVM infrastructure: threads, large heaps

Incremental rewrite of Java applications to Node.js
Building a Universal VM is a Community Effort

• Test your applications with GraalVM
  – Documentation and downloads at http://www.graalvm.org
• Connect your technology with GraalVM
  – Integrate GraalVM into your application
  – Run your own programming language or DSL
  – Build language-agnostic tools
• Join the conversation
  – Report issues or pull requests on GitHub
  – graalvm-users@oss.oracle.com
  – Follow @graalvm