GraalVM

Initialize Once Start Fast: Application Initialization at Build Time

What you should get from this talk?

- Brief knowledge about new technology
- New popular words in your lexicon: Graal and GraalVM
- Passion to try it in your project
- **This is an overview, I haven’t yet used it in Production**

01

GraalVM overview

What is GraalVM

GraalVM – huge **ecosystem** (polyglot VM, compiler, tools and shared runtime) created by Oracle. It can compile and execute different languages on one VM platform.

https://www.graalvm.org/

https://github.com/oracle/graal

Code processing

Languages

- JVM languages (Kotlin, Java, Scala)
- LLVM-based languages like C/C++
- Other: Python, JS, Ruby

GraalVM

Runtime

- Java Hotspot VM
  (normal VM with GC)
- NodeJS environment
  (node signal handlers, e.t.c)
- Standalone native app
  (AOT of JVM bytecode)

transforming interpreter to compiler

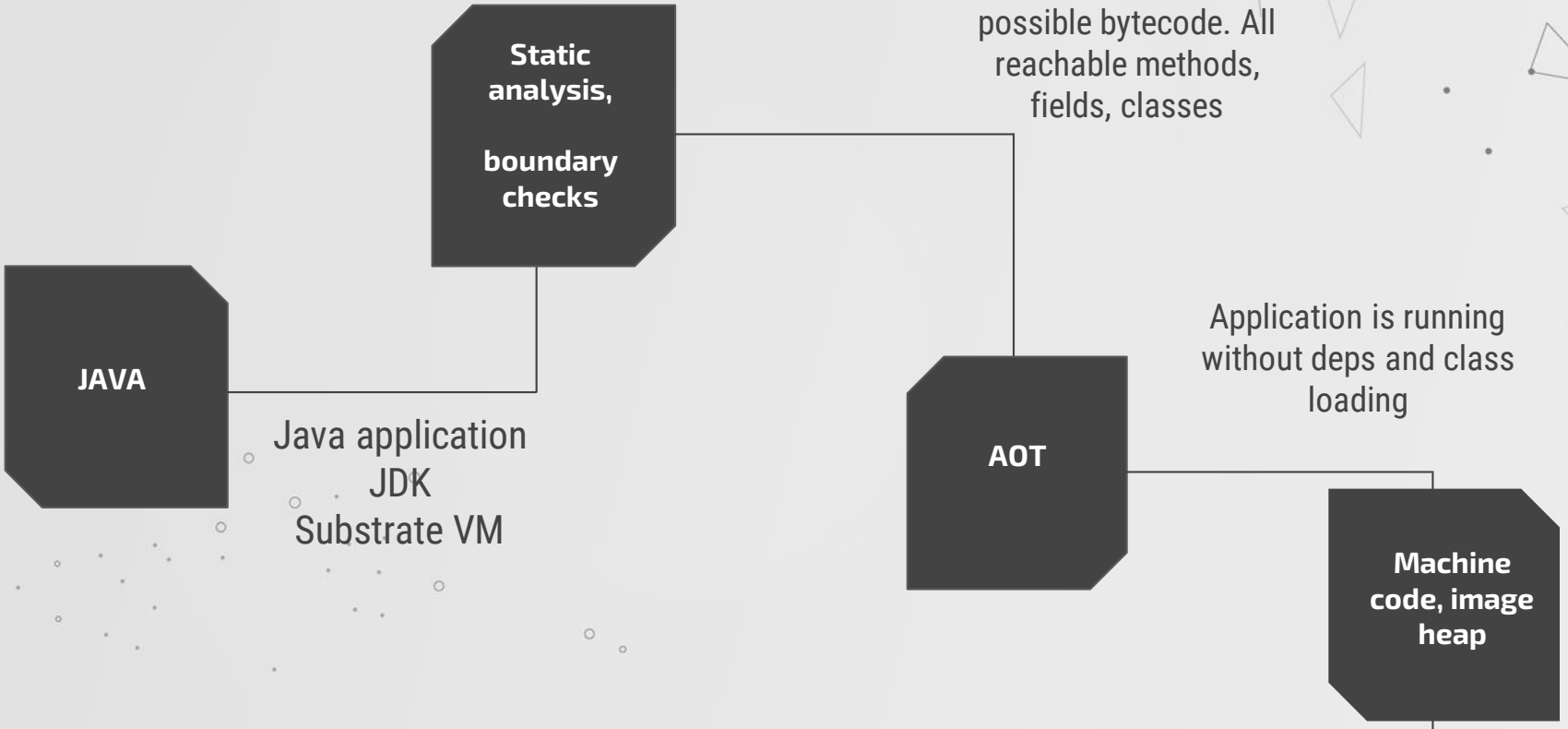
native or managed application

Java in a standalone native app???

- 👍 Ahead of time compilation instead of JIT
- 👍 Application becomes fairly smaller
- 👍 Fast startup (milliseconds for a web server) and low memory usage (less metadata for JIT)
- 👍 Full runtime support including GC

- 👎 Overhead for AOT compilation
- 👎 No portability to platforms
- 👎 No reflection (limited)
- 👎 Should not be used for long running apps

Native execution



Why GraalVM?

- Faster VM and a runtime for Java, Scala, Kotlin, better optimizations than in C2
- Some attempts to have **polyglot** compiler and support to run several non-JVM languages inside of JVM!
- And as a super bonus - native execution of your JVM bytecode with AOT compilation!

Package

akuleshov7@DESKTOP-8I7GFOJ: /usr/lib/jvm/graalvm-ce-java11-20.1.0/bin

```
akuleshov7@DESKTOP-8I7GFOJ:/usr/lib/jvm/graalvm-ce-java11-20.1.0/bin$ ls
gu          javac     jconsole   jfr      jjs     jps          jstack     keytool  npx       rmid
jar         javadoc   jdb        jhsdb    jlink   jrunscript   jstat      lli      pack200   rmiregistry
jarsigner   javap     jdeprscan  jimage   jmap    js           jstatd     node     polyglot  serialver
java        jcmd      jdeps      jinfo    jmod    jshell       jvisualvm  npm      rmic      unpack200
akuleshov7@DESKTOP-8I7GFOJ:/usr/lib/jvm/graalvm-ce-java11-20.1.0/bin$
```

Package

akuleshov7@DESKTOP-8I7GFOJ: /usr/lib/jvm/graalvm-ce-java11-20.1.0/bin

```
akuleshov7@DESKTOP-8I7GFOJ:/usr/lib/jvm/graalvm-ce-java11-20.1.0/bin$ ls
gu          javac     jconsole   jfr      jjs     jps          jstack     keytool  npx       rmid
jar         javadoc   jdb        jhsdb    jlink   jrunscript   jstat      lli      pack200   rmiregistry
jarsigner   javap     jdeprscan  jimage   jmap    js           jstatd     node     polyglot  serialver
java        jcmd      jdeps      jinfo    jmod    jshell       jvisualvm  npm      rmic      unpack200
akuleshov7@DESKTOP-8I7GFOJ:/usr/lib/jvm/graalvm-ce-java11-20.1.0/bin$
```

Which Java JIT compiler is the most optimizing in openJDK?

Package

Which Java JIT compiler is the most optimizing in openJDK?
- **C2 compiler** (also known as ‘opto’).

02

GraalVM architecture

Implementation for JVM-languages

Java, Scala, Kotlin (Normal JVM process runs on hotspot)

GraalVM Compliler
(replaced JIT compilers, default compilers are still in package)

Java Hotspot VM

JVM compiler interface (**JEP253**)

Interpreters

- How do other frameworks work with JVM on non-JVM languages, **Jython** for example?
- Making optimizing compilers themselves, translating language, making mapping from Python to Java bytecode!
- Why not interpreters? – Extremely slow to execute each node of AST. Incredibly dynamic!

Implementation

Python, Ruby, JS, **Haskell (Hi Bulat!)** and some other

Java, Scala, Kotlin

Truffle Framework
(API for creating interpreters on a high level language)

GraalVM Compliler
(optimizes merged interpreter code and app code on runtime)

Java Hotspot VM

Implementation for native languages

C, C++, and

Golang Support #381

Open joegrasse opened this issue on Apr 25, 2018 · 19 comments

Python, Ruby, JS, e.t.c

LLVM bitcode

Java, Scala, Kotlin

Truffle Framework
(API for creating interpreters on a high level language)

GraalVM Compliler
(optimizes merged interpreter code and app code on runtime)

Java Hotspot VM

03

Repository

Some info about setup

Github

oracle / graal

Watch 443 | Star 12.6k | Fork 890

Code | Issues 619 | Pull requests 72 | Actions | Projects 3 | Security 0 | Insights

GraalVM: Run Programs Faster Anywhere https://www.graalvm.org

polyglot vm java javascript python r ruby c

44,692 commits | 15 branches | 0 packages | 155 releases | 173 contributors | View license

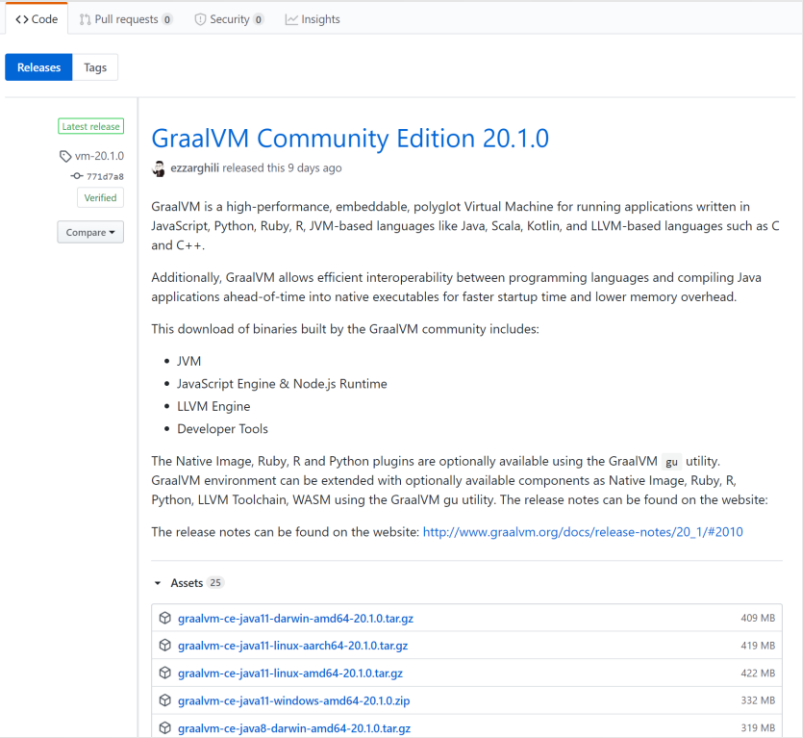
JAVA source code

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

gergo- [GR-21064] Refactoring of System.arraycopy support. ... Latest commit 446bf91 7 hours ago

.github/ISSUE_TEMPLATE	GR-21616 Simplifies the native-image template	3 months ago
ci_includes	Move website from gh-pages branch of Graal GitHub repo to a separate ...	3 days ago
compiler	[GR-21064] Refactoring of System.arraycopy support.	7 hours ago
docs	Add 2019 scientific publications to docs/Publications.md	7 months ago
examples	remove all uses of JDK 11 on SPARC	7 months ago

Github



Code | Pull requests 0 | Security 0 | Insights

Releases | Tags

Latest release

vm-20.1.0

771d7a8

Verified

Compare

GraalVM Community Edition 20.1.0

ezzarghili released this 9 days ago

GraalVM is a high-performance, embeddable, polyglot Virtual Machine for running applications written in JavaScript, Python, Ruby, R, JVM-based languages like Java, Scala, Kotlin, and LLVM-based languages such as C and C++.

Additionally, GraalVM allows efficient interoperability between programming languages and compiling Java applications ahead-of-time into native executables for faster startup time and lower memory overhead.

This download of binaries built by the GraalVM community includes:

- JVM
- JavaScript Engine & Node.js Runtime
- LLVM Engine
- Developer Tools

The Native Image, Ruby, R and Python plugins are optionally available using the GraalVM `gu` utility. GraalVM environment can be extended with optionally available components as Native Image, Ruby, R, Python, LLVM Toolchain, WASM using the GraalVM gu utility. The release notes can be found on the website:

The release notes can be found on the website: http://www.graalvm.org/docs/release-notes/20_1/#2010

Assets 25

graalvm-ce-java11-darwin-amd64-20.1.0.tar.gz	409 MB
graalvm-ce-java11-linux-aarch64-20.1.0.tar.gz	419 MB
graalvm-ce-java11-linux-amd64-20.1.0.tar.gz	422 MB
graalvm-ce-java11-windows-amd64-20.1.0.zip	332 MB
graalvm-ce-java8-darwin-amd64-20.1.0.tar.gz	319 MB

https://github.com/graalvm/graalvm-ce-builds/releases

Simple installation

- Load archive as normal Java
- Set Path and java home:

  $ export PATH=<path to GraalVM>/bin:$PATH

  $ export JAVA_HOME=<path to GraalVM>
- Run java and note that at any time you are able to switch to default JIT compiler:

  $ java –XX:-UseJVMCICompiler

04

Other benefits

Polyglot compiler

- GraalVM allows you to write **polyglot** applications with a seamless way to pass values from one language to another
- It will be run in **ONE** single process
- Now you can write code on the language you prefer **without holywars**!
- If you will use java from Graal then there will be no need to add dependencies for org.graalvm.* - polyglot will be already in classpath

Polyglot compiler

JS goes by the default:

```
import org.graalvm.polyglot.*;

class Polyglot {
    public static void main(String[] args) {
        Context polyglot = Context.create();

        // it can be python or other Truffle language here!!!
        Value array = polyglot.eval("js", "[1,2,42,4]");
        int result = array.getArrayElement(2).asInt();
        System.out.println(result);
    }
}
```

For other languages you will get an error on Runtime:
A language with id 'python' is not installed. Installed languages are: [js]

Polyglot compiler

Need to install python:

$PathToGraal/bin/gu install python

$ PathToGraal/bin/gu install -c org.graalvm.python

akuleshov7@DESKTOP-8I7GFOJ: ~/graal/graalvm-demos/polyglot

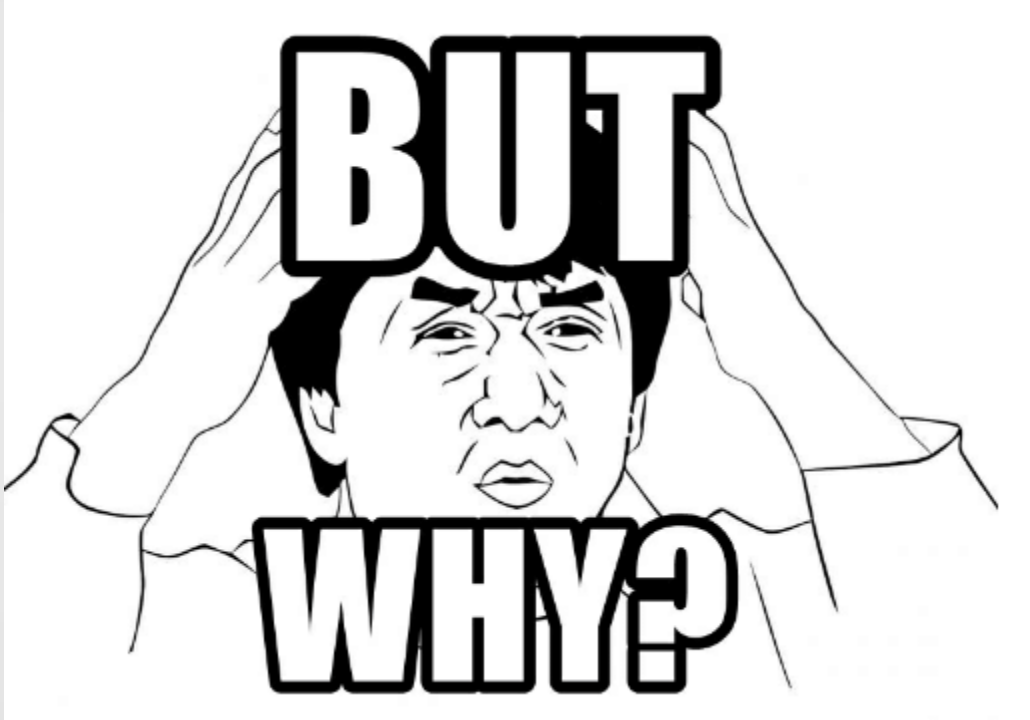
```
package  org.graalvm.demos;
import org.graalvm.polyglot.Context;
import org.graalvm.polyglot.Source;
import org.graalvm.polyglot.Value;

public class HelperTest {
 public static void main(String[] args) {
        Context.Builder builder = Context.newBuilder();
        builder.allowAllAccess(true);
        Context context = builder.build();
        String source = "import polyglot\n" +
                "@polyglot.export_value\n" +
                "def foo(externalInput):\n" +
                "    print('Called with: ' + externalInput)\n" +
                "    return 'Got output'\n\n";

        Source script = Source.create("python", source);
        context.eval(script);
        Value main = context.getPolyglotBindings().getMember("foo");

        Value something = main.execute("myInput");
    }
}
```

Why do I need this hell?
I will never put Ruby inside Java!!



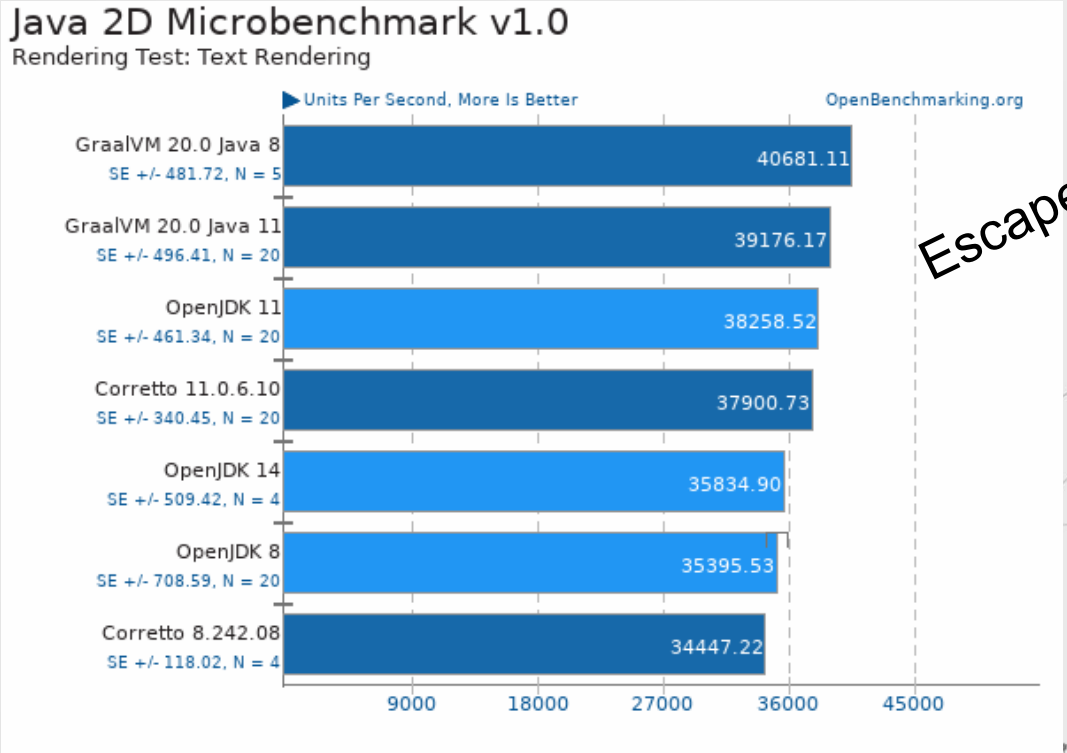
Because…

- Now you can use any JVM tool to watch and control your damn Ruby application!
- You can use for example chromium debugging tools to debug C++
- You can use jconsole or sampler or anything else for Python/Ruby/e.t.c!
- Btw – Ruby is faster on GraalVM ☺

05

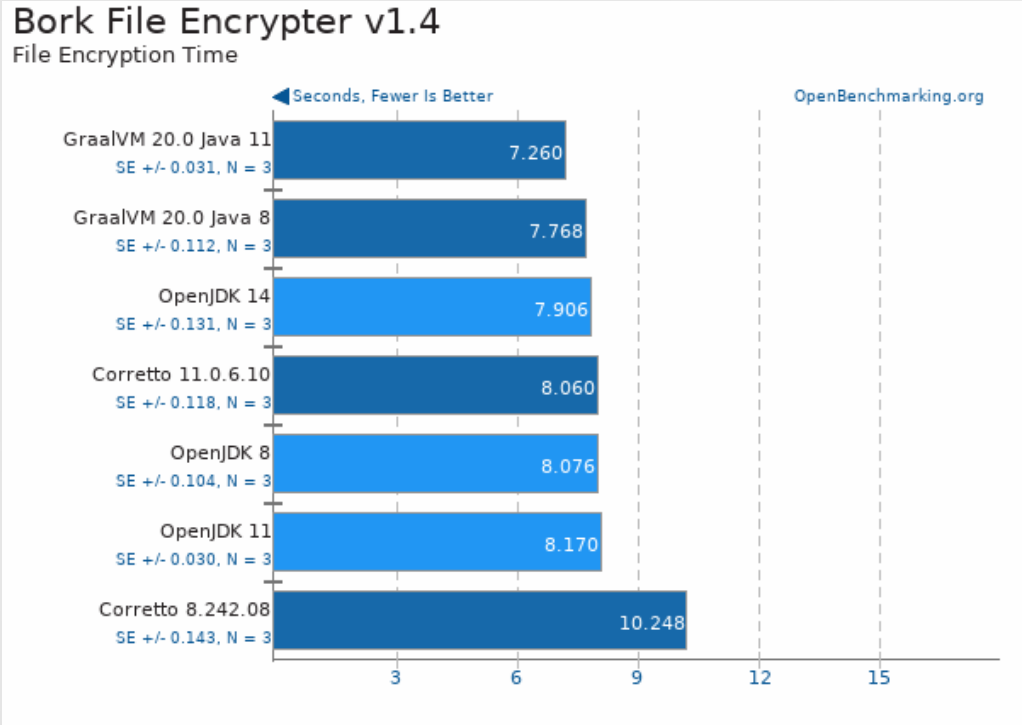
Some benchmarks

Benchmarks

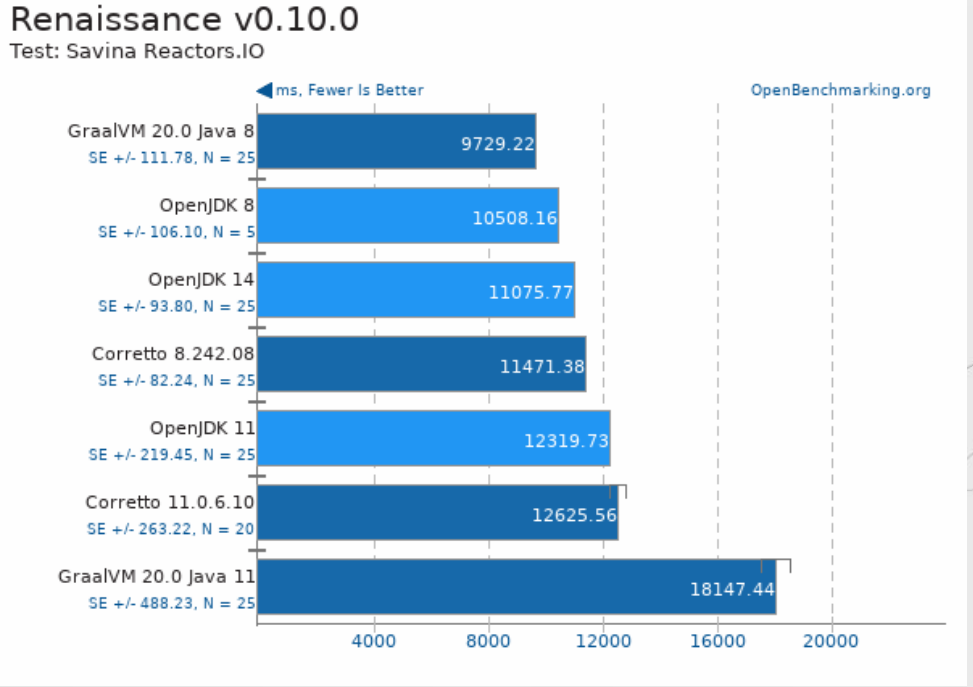


Escape analysis is better?

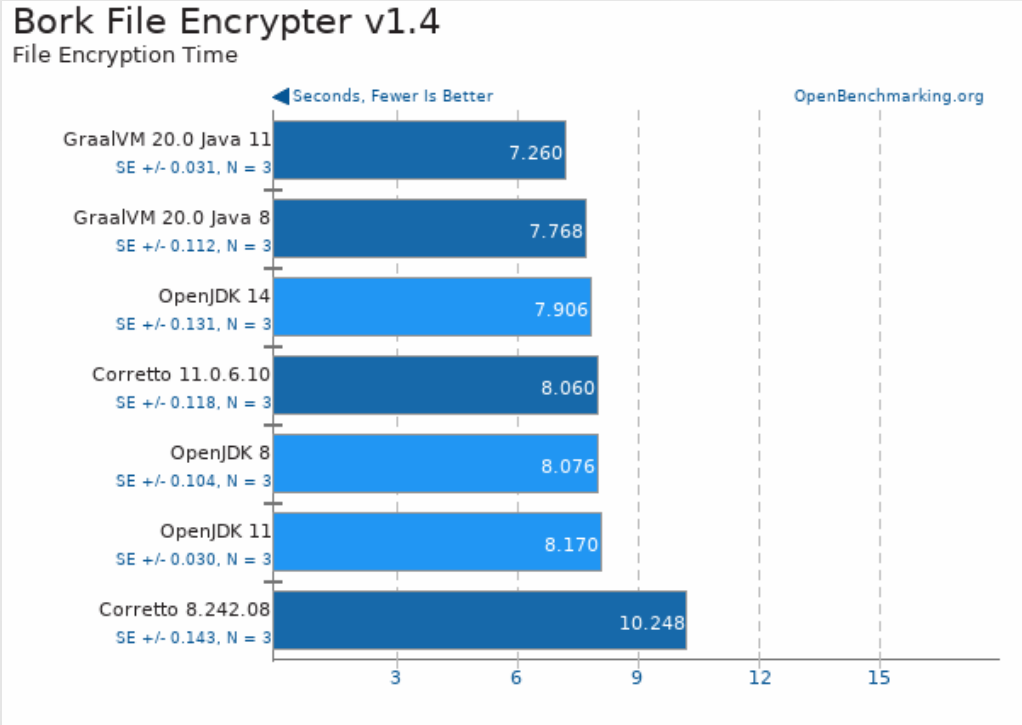
Benchmarks


Bork File Encrypter v1.4
File Encryption Time
Seconds, Fewer Is Better
OpenBenchmarking.org
GraalVM 20.0 Java 11
SE +/- 0.031, N = 3
7.260
GraalVM 20.0 Java 8
SE +/- 0.112, N = 3
7.768
OpenJDK 14
SE +/- 0.131, N = 3
7.906
Corretto 11.0.6.10
SE +/- 0.118, N = 3
8.060
OpenJDK 8
SE +/- 0.104, N = 3
8.076
OpenJDK 11
SE +/- 0.030, N = 3
8.170
Corretto 8.242.08
SE +/- 0.143, N = 3
10.248
3
6
9
12
15

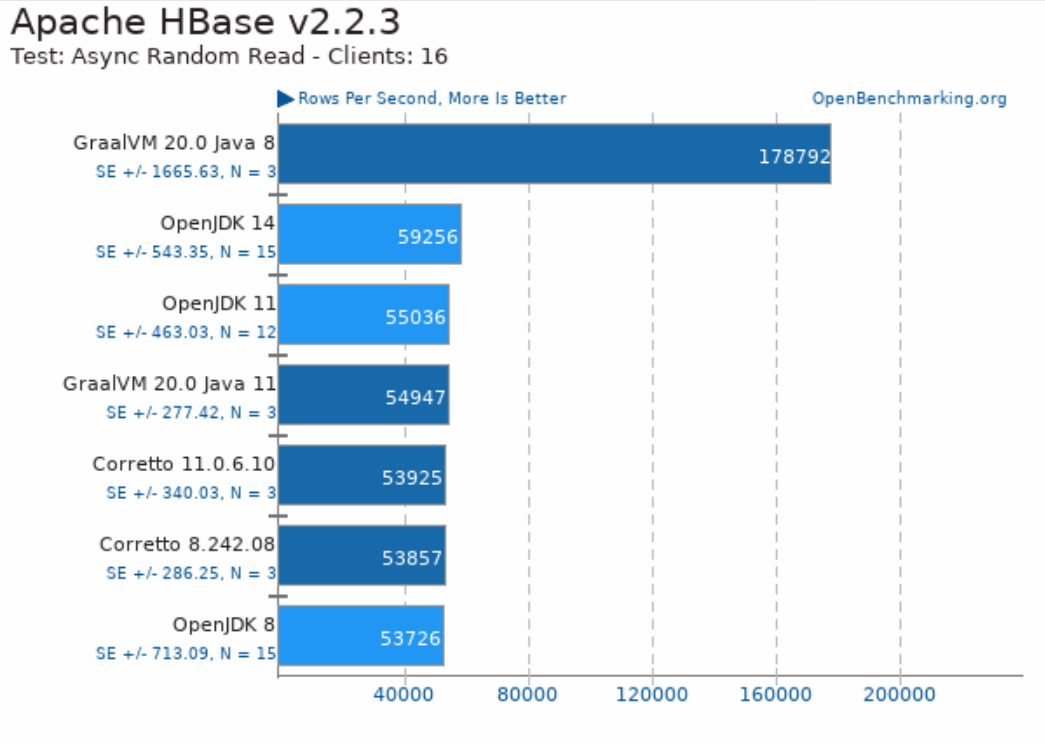
Benchmarks


Renaissance v0.10.0
Test: Savina Reactors.IO
ms, Fewer Is Better
OpenBenchmarking.org
GraalVM 20.0 Java 8
SE +/- 111.78, N = 25
9729.22
OpenJDK 8
SE +/- 106.10, N = 5
10508.16
OpenJDK 14
SE +/- 93.80, N = 25
11075.77
Corretto 8.242.08
SE +/- 82.24, N = 25
11471.38
OpenJDK 11
SE +/- 219.45, N = 25
12319.73
Corretto 11.0.6.10
SE +/- 263.22, N = 20
12625.56
GraalVM 20.0 Java 11
SE +/- 488.23, N = 25
18147.44
4000
8000
12000
16000
20000

Simple installation



Benchmark


Apache HBase v2.2.3
Test: Async Random Read - Clients: 16
Rows Per Second, More Is Better
OpenBenchmarking.org
GraalVM 20.0 Java 8
SE +/- 1665.63, N = 3
178792
OpenJDK 14
SE +/- 543.35, N = 15
59256
OpenJDK 11
SE +/- 463.03, N = 12
55036
GraalVM 20.0 Java 11
SE +/- 277.42, N = 3
54947
Corretto 11.0.6.10
SE +/- 340.03, N = 3
53925
Corretto 8.242.08
SE +/- 286.25, N = 3
53857
OpenJDK 8
SE +/- 713.09, N = 15
53726
40000
80000
120000
160000
200000

Thanks

That’s all folks