Improving GraalVM Reflection File Generation

4/2/22

ntoper@manycore.io + marcus@manycore.io

Introduction

- https://www.magicator.com
- Slice Based Analysis yields correct and complete results
- Despite being a POC, Magicator already resolves instructions GraalVM can't
- A key problem to solve to make GraalVM mainstream
- Outline
  - Slice based analysis
  - State of Reflection in GraalVM
  - How does Magicator work
  - Our results

Magicator

Upload Your Jar

Please drop your Jar, and its Jar dependencies (if any) here. When done, click upload.

Pom Dependencies (optional)

```
<!-- Optionally copy your dependencies tag from your pom.xml to here -->
```

```
Example:
    <dependencies>
        <dependency>
            <groupId>com.google.guava</groupId>
            <artifactId>guava</artifactId>
            <version>29.0</version>
        </dependency>
    </dependencies>
```

Start Upload

Magicator Result

Extraire + magicator_graalvm_config.zip

Emplacement : /META-INF/native-image/

Nom	Taille	Type	Modifié
failed-dynamic-instructions.csv	0 octet	document …	10 octobre 2
jni-config.json	343 octets	document …	10 octobre 2
proxy-config.json	4 octets	document …	10 octobre 2
reflect-config.json	713 octets	document …	10 octobre 2
resource-config.json	37 octets	document …	10 octobre 2
staticizer-info.txt	141 octets	document t…	10 octobre 2

Magicator Result


```
{
  "methods": [{
    "parameterTypes": [],
    "name": "<init>"
  }],
  "name": "com.fasterxml.jackson.databind.ext.Java7HandlersImpl"
},
{
  "methods": [{
    "parameterTypes": [],
    "name": "<init>"
  }],
  "name": "com.fasterxml.jackson.databind.ext.Java7SupportImpl"
},
{"name": "io.manycore.reflection.Meng0"},
{"name": "io.manycore.reflection.Meng1"},
{"name": "io.manycore.reflection.Meng10"},
{"name": "io.manycore.reflection.Meng11"},
{"name": "io.manycore.reflection.Meng12"},
{"name": "io.manycore.reflection.Meng13"},
{"name": "io.manycore.reflection.Meng14"},
{"name": "io.manycore.reflection.Meng15"},
{"name": "io.manycore.reflection.Meng2"},
{"name": "io.manycore.reflection.Meng3"},
{"name": "io.manycore.reflection.Meng4"},
{"name": "io.manycore.reflection.Meng5"},
{"name": "io.manycore.reflection.Meng6"},
{"name": "io.manycore.reflection.Meng7"},
{"name": "io.manycore.reflection.Meng8"},
{"name": "io.manycore.reflection.Meng9"},
{
  "name": "io.micronaut.caffeine.cache.BaseMpscLinkedArrayQueueColdProducerFields",
  "fields": [{"name": "producerLimit"}]
},
{
  "name": "io.micronaut.caffeine.cache.BaseMpscLinkedArrayQueueConsumerFields",
  "fields": [{"name": "consumerIndex"}]
},
{
  "name": "io.micronaut.caffeine.cache.BaseMpscLinkedArrayQueueProducerFields",
  "fields": [{"name": "producerIndex"}]
},
{
  "methods": [{
    "parameterTypes": [],
    "name": "toPath"
  }],
  "name": "java.io.File"
},
```


Micronaut demo app
Extract 184 lines

Slice Based Analysis

- Abstract Interpretation: we execute all instructions with a simplified execution model
- Slice Based Analysis: we execute a few instructions with the real execution model
- This works because:
  - A lot of methods are doing “always the same thing”.
  - e.g. a constant string pushed on the stack in a method
  - >40% of an Android app are amenable to this type of analysis
- Reflection analysis is our first application to “test this question”
  - Used for meta-programmation is often not tied to the program inputs

Is Reflection a Solved Problem?

Program	Number of reflection instructions	Directly substituted by GraalVm
Minecraft Server	9,249	288
Freemind	1,063	144
Mindustry	5,348	377
jEdit	2,022	308
Zookeeper	4,145	198

Completeness & Correctness

```
//classCount is defined somewhere else
Class thisClass =
    Class.forName("io.manycore.reflection.Class" + classCount);
```

We want all the values of thisClass

Native Images Used For Execution Only

- We built Native Images for a specific server instance
- More optimizations available
  - E.g. configuration files are inlined
- It’s similar to how Android is using Dalvik and ART
  - Dalvik used for distribution
  - ART (with AOT) used for execution on the device with specialized code for the device

Almost All Reflection Instructions Can Be Fully Resolved

- False theoretically but true empirically
- Empirically and in “real world programs” all reflection instructions have a well defined set of values
- (hypothesis) It’s because of limit to human cognition
  - We need to limit reflection to “understandable cases”
  - “free reflection” is usually a security problem

Slice Based Analysis Algorithm

- For each reflection instruction
  - Build a backward slice
  - If slice is parameterless: execute it
  - If slice needs parameters,
    - Fetch from call graph all methods invoking the sliced method
    - For each of these methods
      - build a backward slice
    - repeat
  - Remove all slices not linked to an entry point
- For each slice
  - Assemble the slices into a Java program and execute them
  - Gather result values in reflection configuration file

Simple Example

```
for (int methodToCallCount=0; methodToCallCount <= 7; methodToCallCount++) {
    for (int classCount = 0; classCount <= 15; classCount++) {
        Class thisClass = Class.forName("io.manycore.reflection.Show" + classCount);
        System.out.println(thisClass.toString());
    }
}
```

1. Slice

Slice: a new program where this instruction behaves exactly the same as in the original program

```
for (int methodToCallCount=0; methodToCallCount <= 7; methodToCallCount++) {
    for (int classCount = 0; classCount <= 15; classCount++) {
        Class thisClass = Class.forName("io.manycore.reflection.Show" + classCount);
        System.out.println(thisClass.toString());
    }
}
```

2. We execute slice in a JVM with Tracing Agent enabled

EventBus.register(): Original Code

```
public void register(Object subscriber) {
    if (AndroidDependenciesDetector.isAndroidSDKAvailable() &&
        !AndroidDependenciesDetector.areAndroidComponentsAvailable()) {
        // Crash if the user (developer) has not imported the Android compatibility library.
        throw new RuntimeException("It looks like you are using EventBus on Android, " +
                "make sure to add the \"eventbus\" Android library to your dependencies.");
    }


    Class<?> subscriberClass = subscriber.getClass();
    List<SubscriberMethod> subscriberMethods = subscriberMethodFinder.findSubscriberMethods(subscriberClass);
    synchronized (this) {
        for (SubscriberMethod subscriberMethod : subscriberMethods) {
            subscribe(subscriber, subscriberMethod);
        }
    }
}
```

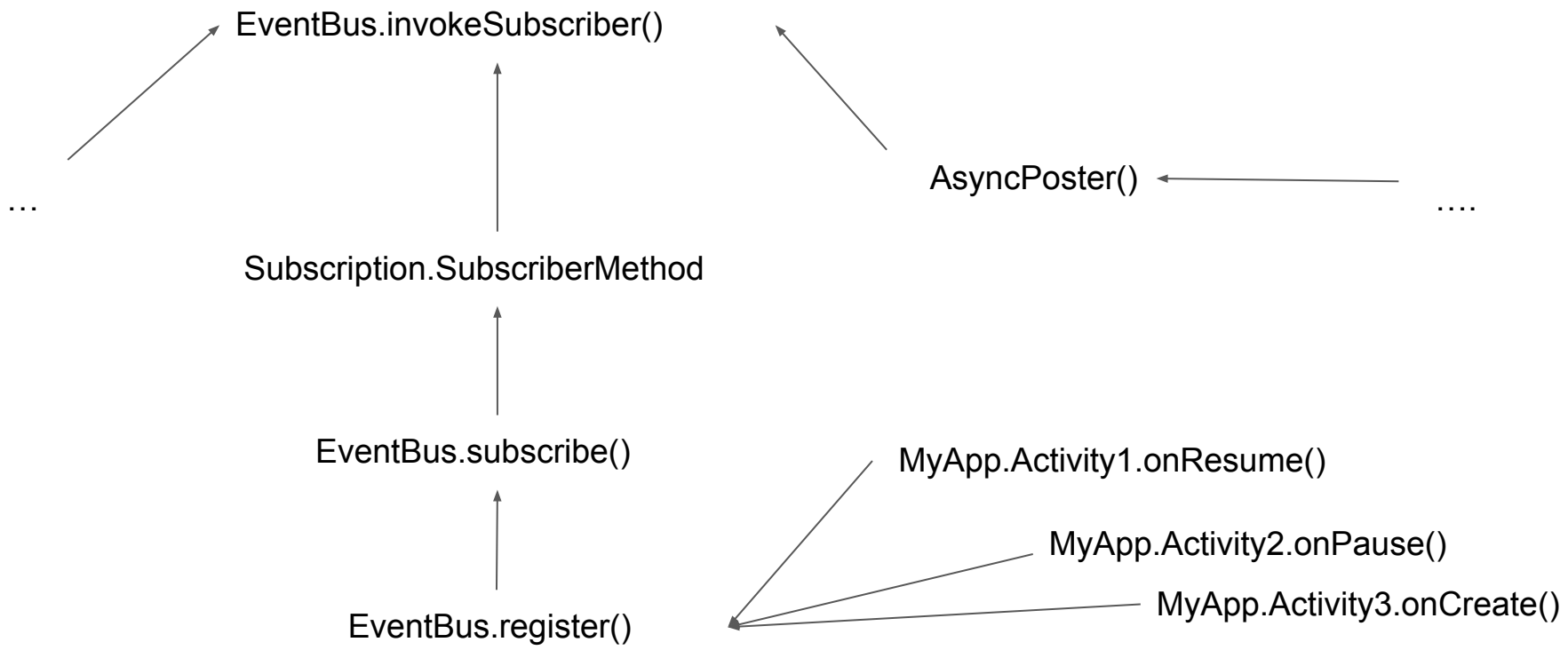
EventBus.register(): Seed

```
public void register(Object subscriber) {
    if (AndroidDependenciesDetector.isAndroidSDKAvailable() &&
        !AndroidDependenciesDetector.areAndroidComponentsAvailable()) {
        // Crash if the user (developer) has not imported the Android compatibility library.
        throw new RuntimeException("It looks like you are using EventBus on Android, " +
                "make sure to add the \"eventbus\" Android library to your dependencies.");
    }


    Class<?> subscriberClass = subscriber.getClass();
    List<SubscriberMethod> subscriberMethods = subscriberMethodFinder.findSubscriberMethods(subscriberClass);
    synchronized (this) {
        for (SubscriberMethod subscriberMethod : subscriberMethods) {
            subscribe(subscriber, subscriberMethod);
        }
    }
}
```

EventBus.register(): Original Code

```
public void register(Object subscriber) {
    if (AndroidDependenciesDetector.isAndroidSDKAvailable() &&
        !AndroidDependenciesDetector.areAndroidComponentsAvailable()) {
        // Crash if the user (developer) has not imported the Android compatibility library.
        throw new RuntimeException("It looks like you are using EventBus on Android, " +
                "make sure to add the \"eventbus\" Android library to your dependencies.");
    }


    Class<?> subscriberClass = subscriber.getClass();
    List<SubscriberMethod> subscriberMethods = subscriberMethodFinder.findSubscriberMethods(subscriberClass);
    synchronized (this) {
        for (SubscriberMethod subscriberMethod : subscriberMethods) {
            subscribe(subscriber, subscriberMethod);
        }
    }
}
```

EventBus CallGraph



The Parameterless Reduced Program

MyApp.Activity1.onResume()

```
EventBus.getDefault().registerSliced(this);
```

EventBus.registerSliced(object Subscriber)

```
Class<?> subscriberClass = subscriber.getClass();
List<SubscriberMethod> subscriberMethods = subscriberMethodFinder.findSubscriberMethods(subscriberClass);
for (SubscriberMethod subscriberMethod : subscriberMethods) {
    subscribe(subscriber, subscriberMethod);
}
```

Tracing Agent and Magicator

Program	**Classes detected by Agent**	**Classes from Magicator not detected by Agent**
Minecraft Server	477	27
Freemind	168	8
Mindustry	289	4
jEdit	139	4
Zookeeper	139	36

Conclusion

- Magicator works already despite being a Proof of Concept
- A key problem to solve for GraalVM
- We can remove all reflection instructions in a program.
  - This is a known source of inefficiency in a program
- We can optimize a program much further using instruction resolution with slice based analysis
  - This “semi-constant” type of instructions needs to be studied more formally
  - More work to be done
- Writing a slicer is much harder than it seems

Roadmap

- N pass resolution to reach all reflection instructions
- Build time/run time static field handling
- Use Espresso instead of an external JVM
- Remove reflection instruction

Questions?

https://www.magicator.com
ntoper@manycore.io
marcus@mancycore.io