ART JIT in Android N

Xueliang ZHONG
Linaro ART Team

linaro-art@linaro.org
Outline

- Android Runtime (ART) and the new challenges
- ART Implementation in Android N
- Tooling
- Performance Data & Findings
- Q & A
ART

- Android Runtime (ART)

- Performance Requirements for ART:
  - Run APPs smoothly.
  - Fast APP startup.
  - Fast APP installation.
  - With minimal overhead.
    - CPU, memory, storage, power, ...
Android KitKat Just-In-Time (JIT) Solution

- Dynamic compilation
Android KitKat Just-In-Time (JIT) Solution

- **APK**
  - Fast Installation

- **INSTALL**
  - Less Storage Required

- **DEX Code**

- **START APP**

- **JIT**
  - JIT compilation overhead
  - Runtime battery overhead

- **Native Code**
Android Marshmallow Ahead-Of-Time (AOT) Solution

- **Static** compilation
Android Marshmallow Ahead-Of-Time (AOT) Solution

APK INSTALL (AOT) Native Code START APP

Slow Installation More Storage Required

Fast startup performance

Fast runtime performance

Less runtime battery consumption
More Challenges for AOT Nowadays

- System Update Time
- Storage Consumption
- Dynamic APK
ART Implementation in Android N

APK > INSTALL > DEX Code > START APP > JIT

Native Code

Compilation Daemon Service > AOT > Native Code > START APP

PROFILE
ART Implementation in Android N

- **JIT Phase**
  - Fast installation
  - Fast system update time (OTA)

[Diagram of ART Implementation process]

1. **APK** → **INSTALL** → **DEX Code** → **START APP** → **JIT** → **Native Code**

- **Compilation Daemon Service** → **AOT** → **Native Code** → **START APP**
ART Implementation in Android N

- Profile Guided AOT:
  - Fast startup and runtime performance
  - Lower overheads: CPU, storage, power ...
ART JIT - Implementation Overview

- JIT Runtime
- JIT Compiler
- Generates Profile File
ART JIT - Implementation Overview

APK > INSTALL > DEX Code > START APP > JIT > Native Code

- **Interpreter**
  - Computed-goto based
  - Opcode handling in ARM assembly

- **Profiling & Triggering JIT Compilation**
  - Based on execution count and heuristics
  - ‘hotness_count’

- **JIT Code Cache**
  - Manages JITed Code
  - Allocating ‘ProfilingInfo’ for hot methods

- **JIT Runtime**
  - JIT Compiler
  - Generates Profile File
ART JIT - Implementation Overview

- Method granularity compilation
- Background compilation.
  - Avoid blocking APP thread (e.g. UI thread)
- Based on ART optimizing compiler.
  - Same compiler used for AOT compilation.
- JIT compilation enhancements in optimizing compiler

- JIT Runtime
- JIT Compiler
- Generates Profile File
ART JIT - Implementation Overview

- ProfileSaver thread
  - Reads ‘ProfilingInfo’ about hot methods
  - Writes such info to disk
- Minimize overhead
  - Wakeup frequency
  - Minimize things to write

- JIT Runtime
- JIT Compiler
- Generates Profile File
ART Hybrid Mode

- Idea of Hybrid Mode (JIT + AOT)
ART Hybrid Mode

- Idea of Hybrid Mode (JIT + AOT)
  - Some users only use part of the APP. Only these frequently used features (and code behind them) are worth compiling into native code.
ART Hybrid Mode

- Idea of Hybrid Mode (JIT + AOT)
  - Some users only use part of the APP. Only frequently used features (and code behind them) are worth compiling into native code.
  - Use JIT phase to find out the frequently used code.
ART Hybrid Mode

- Idea of Hybrid Mode (JIT + AOT)
  - Some users only use part of the APP. Only frequently used features (and code behind them) are worth compiling into native code.
  - Use JIT phase to find out the frequently used code.
  - Use AOT compilation & optimization to speed up frequent use cases.
ART Hybrid Mode

- Idea of Hybrid Mode (JIT + AOT)
  - Some users only use part of the APP. Only frequently used features (and code behind them) are worth compiling into native code.
  - Use JIT phase to find out the frequently used code.
  - Use AOT compilation & optimization to speed up frequent use cases.
  - Avoid paying overheads (compilation, storage, etc) to those less used codes.
ART Hybrid Mode - Overview

- Implementation
  - Generate Offline Profile During JIT
  - Offline Profile File Format
  - Enhanced AOT Compiler (dex2oat)
  - Compilation Daemon Service
**ART Hybrid Mode**

**APK** → **INSTALL** → **DEX Code** → **START APP** → **JIT** → **Native Code**

### Offline Profile File
- Each APP has its own profile.
- Stored in APP local storage.
- Simple format.
- Analysis through **profman** tool.

```plaintext
base.apk
  methods: 1,2,5
  classes: 0,6,13
base.apk:classes2.dex
  methods: 5,12,28
  classes: 17,19,33
```
ART Hybrid Mode

- Compilation Daemon
  - Triggers AOT when the device is charging and idle.
  - ‘BackgroundDexOptService’
  - frameworks/base/services/core/java/com/android/server/pm/
ART Hybrid Mode

Profile Guided AOT

- dex2oat enhanced to accept profile
- selective compilation based on profile

Profile:
- dex2oat32 | compiler_driver.cc Skipped method: void MyClass.<init>()
- dex2oat32 | compiler_driver.cc Compiled method: int MyClass.foo(int)
- dex2oat32 | compiler_driver.cc Compiled method: void MyClass.main()
- dex2oat32 | compiler_driver.cc Skipped method: boolean MyClass.verify()
- dex2oat32 | compiler_driver.cc Skipped method: void Class2.<init>()
ART Hybrid Mode

- Run in Hybrid Mode
  - All previous ‘hot’ methods are AOT compiled
  - JIT compiler is still loaded in memory
    - For unexplored features/codes.
    - May generate new profiles.
Tools

- TraceView (Android Studio / Android SDK)
Tools

- **Systrace (Android Studio / Android SDK)**
  - Enable 'dalvik' trace
  - Analyze JIT behaviors: JIT code cache, ProfilerSaver, GC, etc.
  - Helps to measure:
    - i. Invoke Time (1, 5, 6)
    - ii. JIT compilation time for each method (2, 3)
    - iii. JITed code execution (4)
Tools

- JIT Compiler Analysis
  - CFG dump
    
    ```
    dalvikvm -Xusejit:true -Xcompiler-option --dump-cfg=jit.cfg -cp Main.dex Main
    ```
  
  - Profiling JITed code
    - Support generating `/tmp/perf-PID-maps` for method level profiling with `perf`.
    - Profile AOT compiled code is preferred, for instruction level profiling.
  
  - Other useful JIT statistics & logs:
    - `-verbose:compiler, -verbose:jit`
    - `-XX:DumpJITInfoOnShutdown`
Tools

- **Profile Guided AOT Analysis - profman**

  ```
  profman --profile-file=<filename> --dump-only
  This command can dump the contents in the given prof file.
  
  profman --generate-test-profile=<filename>
  This command can generate a random profile file for testing.
  ```

  **runtime/jit/offline_profiling_info.cc**
  // Debug flag to ignore checksums when testing.
  // Used to facilitate testing across a large number of apps.
  static constexpr bool kDebugIgnoreChecksum = true;

  **compiler/driver/compiler_driver.cc**
  // Print additional info during profile guided compilation.
  static constexpr bool kDebugProfileGuidedCompilation = true;
Performance & Findings

- A Popular News Reader APP

<table>
<thead>
<tr>
<th></th>
<th>APP Installation Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full AOT Approach</td>
<td>88.7 sec</td>
</tr>
<tr>
<td>Hybrid Mode</td>
<td>11.4 sec</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>APP Startup Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full AOT Approach</td>
<td>2.4 sec</td>
</tr>
<tr>
<td>Hybrid Mode (JIT)</td>
<td>2.9 sec</td>
</tr>
</tbody>
</table>
Performance & Findings

● A Popular News Reader APP

<table>
<thead>
<tr>
<th></th>
<th>.oat File Size in APP Local Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full AOT Approach</td>
<td>97 MB</td>
</tr>
<tr>
<td>Hybrid Mode</td>
<td>42 MB (.prof file size 0.03MB)</td>
</tr>
</tbody>
</table>
Performance & Findings

- Micro Benchmark Performance, AOT vs. JIT
More Talks & Resources

● Linaro-ART Team:
  ○ VIXL: A Programmatic Assembler and Disassembler for AArch32
  ○ Android Runtime Performance Analysis

● The Evolution of ART - Google I/O 2016
  ○ https://www.youtube.com/watch?v=fwMM6g7wpQ8

● Implementing ART Just-In-Time (JIT) Compiler
Thank You

#LAS16
For further information: www.linaro.org
LAS16 keynotes and videos on: connect.linaro.org
Backup
Performance & Findings - Example

- Micro Benchmarking – Running Unexpected Modes

```java
void runAlgorithm(int iterations) {
    for(i=0; i<iterations; i++) {
        // long operations ... 
    }
}

void main() {
    int iterations = 20000;
    int before = time();
    runAlgorithm(iterations);
    int after = time();
    // ... 
}
```
Performance & Findings - Example

- JIT’s Advantage over AOT – Inlining

```java
public class MyClass {
    public /*private*/ /*final*/ /*static*/ int arithmeticSeries(int i) {
        if(i == 0)
            return 0;
        else
            return i + arithmeticSeries(i - 1); // invoke-virtual
    }
}
```
ART JIT - Workflow

DEX file

art::dalvikvm
android::AndroidRuntime

ART runtime: Execute method

is it compiled?

no

interpretation

no

AddSamples

hit jit threshold?

no

JIT compiler thread

yes

optimizing compiler

hit warm threshold?

yes

JIT creates ProfilingInfo

ProfileSaver thread

gets ProfilingInfo

saves

commit code

.jit file

jit code cache