Android Optimizing Compiler

New Member Assimilation Guide
Welcome

- Scott Wakeling
  - ARM Cambridge
  - Linaro ART Team member
- The aim of this talk:
  - Guide new and potential members
    - How we work
    - Optimizing compiler
  - Share what we do
  - Invite discussion
Agenda

- Linaro ART Team
- Static Analysis
- Dynamic Analysis
- Testing
- Debugging
- Benchmarking
- Private and Upstream Review
Linaro ART Team
Who We Are

- Linaro Android Runtime (ART) Team
- Grew out of Android 64-bit port
- Subset of Linaro Mobile Group (LMG)
- 7 engineers @ ARM Cambridge, UK
- 2 engineers @ Spreadtrum, China
What We Do

- Contribute to AOSP master branch (Tip)
- Optimizing compiler
  - Dex bytecode
  - Intermediate Representation (IR)
  - ARM/ARM64
- R&D
  - Prefetch / Caching Optimizations / SIMD Vectorization
  - ARM64 simulation on foreign architectures
- Member only optimizations (Stable)
What We Have Done

- Optimizing compiler ARM64 port
  - JNI compiler, VIXL
- VM/Runtime improvements
- Intrinsics
- Instruction Simplification
- Instruction Scheduling
- New IRs
- Platform specific optimizations
How We Do It

- IRC, Email, Video Conferencing
- Weekly 'ART Class'
  - Progress updates
  - Slides (Android, ART, Compiler Theory, etc.)
- Private Review
  - Linaro Gerrit / Jenkins
- Public Review
  - Upstreamed?
Static Analysis
Analysis (IR)

dex2oat --dump-cfg=foo.cfg

- Control Flow Graph (.cfg)
  - Per method, per compilation pass
  - Basic Blocks -> IR -> disassembly

- IRHydra
  - http://mrale.ph/irhydra/2

- C1Visualizer
  - https://java.net/projects/c1visualizer
Analysis (IR / C1Visualizer)

```java
static long rotateRight(long value, int shift) {
    return (value >>> shift) | (value << (64 - shift));
}
```

```
14  | B1  <- B0  -> B2  dom B0 [-1, -1]
15  | Locals size 0 [None]
16  | p_bci_use_tid_instruction
17  | 0  1  j13  UShr [j7,i9]
18  | 0  1  i17  Sub [i16,i9]
19  | 0  1  j21  Shl [j7,i17]
20  | 0  1  j25  Or [j13,j21]
21  | 0  0  v28  Return [j25]
```
Analysis (.oat / oatdump)

```
oatdump --oat-file=foo.oat --output=foo.dump
```

- **boot.oat**
  - Core Java libraries
  - Android Frameworks
- **Unexpected changes (good or bad)**
- **How to diff 1.35Gb of output?**
  - [https://android-review.googlesource.com/#/c/101373](https://android-review.googlesource.com/#/c/101373)
  - `./split-oatdump.py before.dump before/`
  - `./split-oatdump.py after.dump after/`
  - meld before after
Analysis (.oat / meld)
Analysis (.oat / meld)
Dynamic Analysis
Analysis (perf)

  
  ```
  dx --dex --output=Foo.dex Foo.class
  adb shell dalvikvm -cp /data/local/tmp/Foo.dex Foo
  adb shell perf record -g -o /data/local/tmp/perf.data dalvikvm [...]
  perf report
  ```

- Performance counters

  ```
  perf stat -e L1-dcache-load-misses,L1-dcache-loads ./prfm
  Performance counter stats for './prfm':
  259655289 L1-dcache-load-misses # 13.22% of all L1-dcache hits
  1964277523 L1-dcache-loads
  45.634398056 seconds time elapsed
  ```
Analysis (StreamLine)

- ARM Streamline 5.23
  - Performance counters
  - Process/task tracing
  - Timeline view (samples, CPU, Memory, etc.)
  - Function view
  - Call path view
  - Streamline Annotations
  - etc.
Analysis (StreamLine / Capture)

- Running a Capture Session on your device
- prebuilt gator daemon
  
  ```
  adb push $DS-5/sw/streamline/bin/arm/gatord /data/gatord
  adb shell './data/gatord &'
  adb forward tcp:8080 tcp:8080
  ```

- ‘Connection Address’
  - “localhost” if you are port forwarding

- For debug symbols in the output
  - Add an image filepath to ‘Program Images’
Analysis (StreamLine / Counters)

● Timeline View
  ○ Default list of pre-configured performance counters
  ○ Cache, Branch, Fault, Interrupts, IO, Memory etc.

● Counter Configuration
  ○ Cortex-A53, etc.
    ■ Branch mis-predicts
    ■ L1 inst access
    ■ L2 data access
Analysis (StreamLine / Timeline)

- **CPU Activity**
  - User: 100%
  - System

- **Bus**
  - Access: 800 K

- **Cache**
  - Data refill: 200 K
  - Data TLB refill
  - Linefill

- **Clock**
  - Cycles: 200 MHz

- **Instruction**
  - Executed: 20 M
## Analysis (StreamLine / Disassembly)

```c
int main(int argc, char** argv) {
    buffer = malloc(NUM ARRAYS * sizeof(int*));
    for (i = 0; i < NUM ARRAYS; i++) {
        buffer[i] = malloc(BUF SIZE);
        for (j = 0; j < BUF SIZE / sizeof(int); j++) {
            buffer[i][j] = i;
        }
    }
}
```

<table>
<thead>
<tr>
<th>Samples</th>
<th>% Samples</th>
<th>Address</th>
<th>Opcode</th>
<th>Disassembly</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6.25%</td>
<td>0x00400B88</td>
<td>ADD</td>
<td>x1,x19,#0xc30</td>
</tr>
<tr>
<td>1</td>
<td>6.25%</td>
<td>0x00400B8C</td>
<td>LDR</td>
<td>w2,[x1,#8]</td>
</tr>
<tr>
<td>3</td>
<td>18.75%</td>
<td>0x00400B90</td>
<td>LDR</td>
<td>x3,[x19,#0xc30]</td>
</tr>
<tr>
<td>6</td>
<td>37.50%</td>
<td>0x00400B94</td>
<td>LDR</td>
<td>x3,[x3,w2,SXTW #3]</td>
</tr>
<tr>
<td>1</td>
<td>6.25%</td>
<td>0x00400B98</td>
<td>STR</td>
<td>w2,[x3,x0,LSL #2]</td>
</tr>
<tr>
<td>1</td>
<td>6.25%</td>
<td>0x00400B9C</td>
<td>LDR</td>
<td>w0,[x1,#0xc]</td>
</tr>
<tr>
<td>2</td>
<td>12.50%</td>
<td>0x00400BA0</td>
<td>ADD</td>
<td>w0,w0,#1</td>
</tr>
</tbody>
</table>
Testing
Testing (Automation)

- ART Java tests (art/test)
  - Host, Target (inc. libcore)
- gTests
- IR Checker
- Benchmarking
Testing (Manual)

- Automation is always WIP
- Tip - “always changing, often broken”
- ~ 1.5 billion Android devices worldwide
- ART Target Tests
  
  ART_TEST_OPTIMIZING=true m test-art-target-run-test-optimizing -j40

- gTests
  
  mma -j50 test-art-gtest

- libcore
  
  make core-tests jsr166-tests vogar
  art/tools/run-libcore-tests.sh
Testing (IR Checker Tests)

```java
public static int rotateRight(int value, int distance) {
    return (value >>> distance) | (value << (32 - distance));
}
```
Debugging
Debugging (gdb)

- **gdb on target device**
  - `aarch64-linux-android-gdb`
    
    ```
    $ ./run-test --gdb --64 557-my-art-test
    $ adb forward tcp:5039 tcp:5039
    $ aarch64-linux-android-gdb $OUT/symbols/system/bin/dalvikvm64
    ```
  - `gdbclient`
    
    ```
    $ adb shell gdbserver :5039 /system/bin/my_test_app $ARGS
    $ Listening on port 5039
    $ gdbclient <my_test_app pid>
    ```
Debugging (gdb / oat methods)

--- a/compiler/optimizing/code_generator_arm64.cc
+++ b/compiler/optimizing/code_generator_arm64.cc

@@ -577,6 +577,8 @@ void CodeGeneratorARM64::GenerateFrameEntry() {
     BlockPoolsScope block_pools(masm);
     __ Bind(&frame_entry_label_);
     + __ Brk(0);
Debugging (Red / Green)

- Tip is *in development* by definition
- Stepping through code is valuable
  - even when things appear to work
  - even when disassembly looks right
- Green light
  - necessary but not sufficient
- Red light
  - helps prove intended behaviour
Benchmarking
Benchmarking

- What performance benefit does this give us?
  - Both private review and upstream
- No regressions(?)
- Benchmark history (Jenkins)
  - Track progress and set future targets
  - Comparisons
    - 32 / 64-bit
    - Member Stable / Baseline Stable
Benchmarking

- ART Performance Test Framework
  - cmdline, boot-to-gui not required (Juno)
  - Automated reports and graphing
    - [https://art-reports.linaro.org/](https://art-reports.linaro.org/)
  - Peak memory usage, oat size etc.
  - Added CaffeineMark, BenchmarksGame suites
  - ~60 benchmarks with multiple profile points

- Get access, write benchmarks
  - [https://goo.gl/OKCK5D](https://goo.gl/OKCK5D)
Private and Upstream Review
Private Review

- Do exactly one thing, as per commit message
- Passes tests
  - Relative to current HEAD, anyway!
- Recompile boot.oat and reboot
- “Improve something, regress nothing”
- Android Code Style Guidelines
- You believe it is ready for upstream
  - Otherwise, WIP or RFC
Upstream Review

- Impact on other platforms
- Clear deviations from existing code
- DCHECK / UNREACHABLE
  - tests assumptions and “shoulds”
- Insufficient test coverage
- Ongoing upstream reviews are priority
  - git rebase..
The Future?

- General compiler work
- Code generation
- Caching optimizations
- Simulator
- SIMD Vectorization
- Something else?*

*Requires upstream appetite
Resources

- ARTNewStarter
  - https://wiki.linaro.org/Internal/ARTIntroduction

- LMG Reference Library
  - https://wiki.linaro.org/LMG/Engineering

- Linaro Google Drive
  - https://goo.gl/m9Zqcu

- Android Code Style Guidelines

- DS-5 Android Performance Analysis Tutorial
  - http://goo.gl/V2vOGy

- split-oatdump.py
  - https://android-review.googlesource.com/#/c/101373
Thank you

scott.wakeling@linaro.org