SAN19-214 Deploying ARMv8.x Features On Android Runtime

Xueliang Zhong
September 2019
The Android Runtime (ART) is the main application runtime environment layer used by the Android operating system, which supports Android framework code and Android Apps written in Java/Kotlin.
LCG ART Team & Our Contributions To AOSP/ART

- **Who are we:**
  - Subset of Linaro LCG.
  - Focusing on optimising ART for Arm/Arm64 platforms.

- **Our Contributions to AOSP/ART:**
  - Enhanced ART compiler auto-vectorization (w/ SIMD) & loop body copying framework.
  - Introduced instruction scheduling phase in ART compiler, to improve Arm b.L platforms.
  - Improve intrinsics for ART Arm/Arm64 backends.
  - Improve instruction simplifier for ART compiler Arm/Arm64 backends.
  - Enhanced ART JIT profiling & analysis (perf inject).
  - Maintain AOSP/VIXL assembler/disassembler.
  - Contributed various tests and benchmarks.
In 2018, flagship Android phones with ARMv8.2 CPU features (Google Pixel3, Samsung Galaxy S9, etc) started shipping in volume.

The ARMv8.1 and ARMv8.2 instruction sets have introduced several enhancements to AArch64 atomic read-write instructions, additions to the Advanced SIMD instruction set, half-precision floating point data processing support, memory model enhancements, introduction of RAS support, and introduction of statistical profiling, etc. As well as the additions, the optional CRC instructions in v8.0 become a requirement in ARMv8.1.

Get Android Software Stack Ready for ARMv8.2

As the Android Runtime layer sits quite high in Android software stack, introducing new instructions in ART requires Android SW stack being ready for ARMv8.2.

- Why? - ART CPU feature detection
  - Dynamic detection (HWCAP):
    - getauxval(AT_HWCAP) && HWCAP_FPHP == 1 ?
  - Static detection (ACLE macros):
    - #ifdef __ARM_FEATURE_DOTPROD

- Android common kernel (4.9 and 4.14)

- Bionic C library
  - https://android-review.googlesource.com/c/platform/bionic/+//605504

- AOSP prebuilt Clang compiler
  - https://android.googlesource.com/platform/prebuilts/clang/host/linux-x86/
Get ART Ready for ARMv8.2

- ARMv8.x CPU feature detection in ART.
  - https://android-review.googlesource.com/c/platform/art/+/765542

- VIXL ARMv8.3 support (assembler support)
  - https://android-review.googlesource.com/c/platform/external/vixl/+/801539
ARMv8.0-CRC32

- Speed up JAVA APIs on ART:
  - `java.util.zip.CRC32.update(int b)`
  - `java.util.zip.CRC32.update(byte[] b)`
  - `java.util.zip.CRC32.update(byte[] b, int off, int len)`
  - `java.util.zip.CRC32.update(ByteBuffer buffer)`

- CRC32 intrinsic, `CRC32.update(int b)` example:
  ```java
  MVN W0, W2 ; W2 == b
  CRC32B W0, W0, W1 ; W1 == this.crc
  MVN W0, W0 ; return W0
  ```

- Merged upstream:
  - [https://android-review.googlesource.com/c/platform/art/+/840871](https://android-review.googlesource.com/c/platform/art/+/840871)
  - [https://android-review.googlesource.com/c/platform/art/+/810681](https://android-review.googlesource.com/c/platform/art/+/810681)
ARMv8.2-FP16

- Speed up Android Half class APIs
  - `android.util.Half.toFloat(short h)`
  - `android.util.Half.toHalf(float f)`
  - `android.util.Half.floor()`
  - `android.util.Half.ceil()`
  - `android.util.Half.round()`
  - ...

- FP16 intrinsic, `toFloat()` example
  - `public static float toFloat(short h) {  // Java Implementation`
  - `int bits = h & 0xffff;`
  - `int s = bits & SIGN_MASK;`
  - `int e = (bits >>> EXPONENT_SHIFT) & SHIFTED_EXPONENT_MASK;`
  - `int m = (bits ) & SIGNIFICAND_MASK;`
  - `int outE = 0;`
  - `int outM = 0;`
  - `if (e == 0) {  // Denormal or 0   if (m != 0) {
    // Convert denorm fp16 into normalized fp32
    float o = Float.intBitsToFloat(FP32_DENORMAL_MAGIC + m);
    o -= FP32_DENORMAL_FLOAT;
    return s == 0 ? o : -o;`
  } else {
    outM = m << 13;
    if (e == 0x1f) {  // Infinite or NaN
      outE = 0xff;
      if (outM != 0) {  // SNaNs are quieted
        outM |= FP32_QNAN_MASK;
      }
    } else {
      outE = e - EXPONENT_BIAS + FP32_EXPONENT_BIAS;
    }
  }
  int out = (s << 16) | (outE << FP32_EXPONENT_SHIFT) | outM;
  return Float.intBitsToFloat(out);`

- Also, fixed/improved Android Half code:
  - SNaN handling
  - Half.round documentation

- Merged upstream:
  - `https://android-review.googlesource.com/c/platform/art/+/845944`

FMOV H31, W1    ; W1 == h
FCVT S0, H31    ; return S0
### ARMv8.2-DotProd

- Optimise Java Dot Product Loop Code with SIMD

<table>
<thead>
<tr>
<th>Java Dot Product Loop Code</th>
<th>Optimise Loop SIMD UDOT</th>
</tr>
</thead>
<tbody>
<tr>
<td>// byte[] a; byte[] b; for (int i = 0; i &lt; size; i++) {</td>
<td>...</td>
</tr>
<tr>
<td>sum += a[i] * b[i];</td>
<td>for (int i = 0; i &lt; size; i+=16)</td>
</tr>
<tr>
<td>}</td>
<td>UDOT v0.4s, v1.16b, v2.16b</td>
</tr>
</tbody>
</table>

- Merged upstream:
  - [https://android-review.googlesource.com/c/platform/art/+/740301](https://android-review.googlesource.com/c/platform/art/+/740301)
Future Work

● Integrate VIXL Simulator in ART:
  ○ The VIXL simulator allows developers to run or single-step compiler generated code, and view registers and memory.
  ○ This work will improve ARMv8.x instruction debugging & testing in ART.
    ■ ART developers can debug ART generated ARMv8.x code on host machines.
    ■ Test ART ARMv8.x features without connecting to a real Android device.

● Optimise ART compiler with more ARMv8.x instructions
  ○ e.g. SVE, FP16, etc.
Thank you

Join Linaro to accelerate deployment of your Arm-based solutions through collaboration

contactus@linaro.org