Porting a 32-bit OS to AArch64

David Butcher
Principal Engineer

Why Android?

- Full Consumer OS with source available
- Meeting point of several interesting technologies
  - Dalvik VM & JIT
  - Google V8 JavaScript JIT
  - UI rendered using OpenGL ES
- Meeting point of form factors
  - Phone, Tablet, 'laptop-lite'
- After porting Linux - the next logical choice

The Use of Models

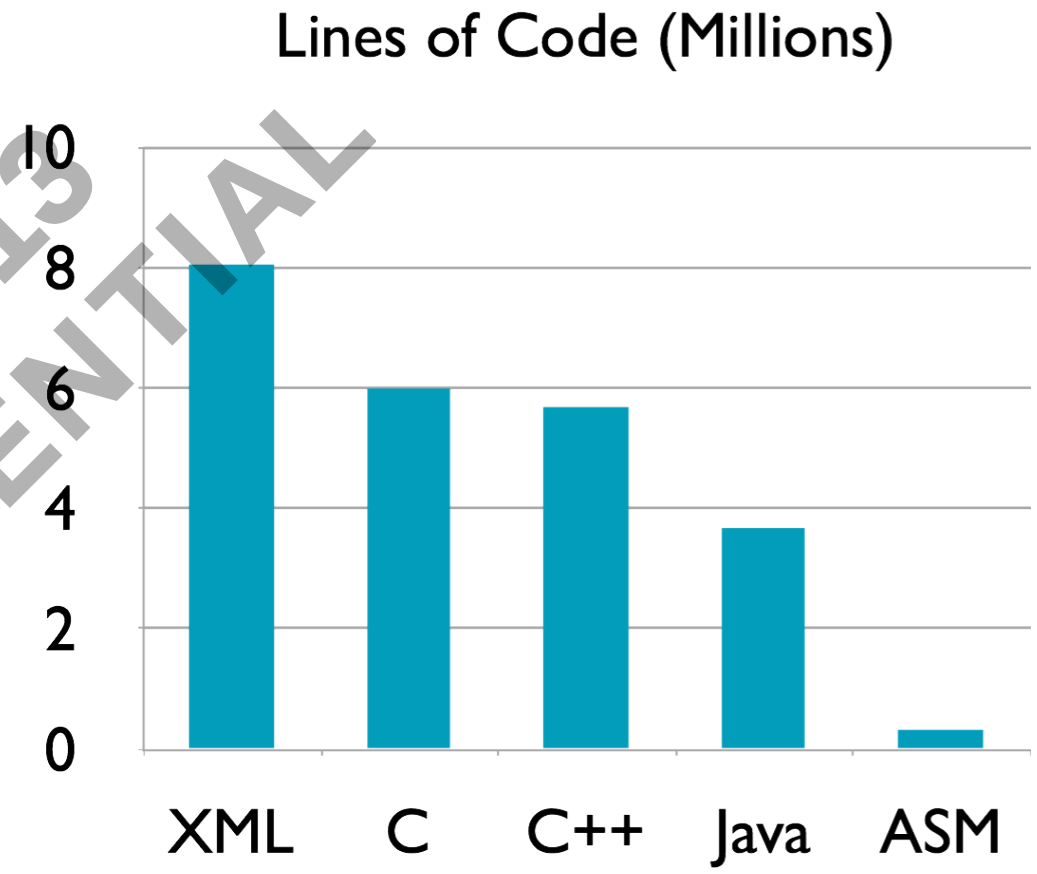
- Porting takes time
- Optimizing takes time

- Approach:
  - Functional Port on models
  - Port existing optimized ARM or Thumb2 code
  - Optimize as soon as hardware is available

  - Port to the Architecture
  - Optimize for the Product

How Big is this Job?

- Android
  - 330+ Separate Projects
  - 24 Million Physical Source Lines of Code (SLOC)
- Daunting?
  - 64-bit porting well understood
  - Compilers can do a lot
- Not so much
  - 78% of projects so far have needed no changes
  - 16% of projects require minor type/casting changes
  - 3% of projects require implementation changes
  - 3% of projects are trickier



Changes for 64-bit

- Going from ILP32 to LP64
  - longs go from 32 to 64 bits (from 4 to 8 bytes)
  - pointers make the same transition
  - long longs are unchanged (8 bytes)
- Use sizeof() instead of a constant
  - `(void **) calloc(4, 100);` should be `(void**) calloc(sizeof(void *), 100);`
- Use #include <stdint.h> and the types defined there
  - uint64_t, uintptr_t etc.
- Postfix literals that might be 32 or 64bit with L (eg 0x0041524dL , 1L)
  - With literals that will be 64-bit in both cases use LL
- Do not burn bridges, you will want to build the same code for 32-bit
  - Use __LP64 where behaviour must be different

Kernel Porting

- Ashmem driver in Android kernel
  - Need 64-bit support before adding compatibility for 32-bit on an AArch64 kernel
  - 27-line patch to ~850 lines across 2 files (ashmem.c and ashmem.h) - already upstreamed
  - Limit on sharing is still 4GB - simple patch required for more but is it needed?
- Binder driver in Android kernel
  - Support for 64-bit filesystems upstreamed
  - 32-bit filesystem on 64-bit kernel patches in final review
- Changes are reasonably simple
  - Patch review process between ARM, Google and Linux Kernel engineers was thorough
  - Final patches a good fit for everyones needs
  - Will be able to boot 64-bit Android on a 3.12 mainline kernel
  - Still SoC specific efforts required for final products

Kernel Porting: Example Patch

```
+/* support of 32bit userspace on 64bit platforms */
+#ifdef CONFIG_COMPAT
+static long compat_ashmem_ioctl(struct file *file, unsigned int cmd, unsigned long arg)
+{
+
+       switch (cmd) {
+       case COMPAT_ASHMEM_SET_SIZE:
+               cmd = ASHMEM_SET_SIZE;
+               break;
+       case COMPAT_ASHMEM_SET_PROT_MASK:
+               cmd = ASHMEM_SET_PROT_MASK;
+               break;
+       }
+       return ashmem_ioctl(file, cmd, arg);
+}
+#endif
+
static const struct file_operations ashmem_fops = {
         .owner = THIS_MODULE,
@@ -710,7 +727,9 @@ static const struct file_operations ashmem_fops = {
-       .compat_ioctl = ashmem_ioctl,
+#ifdef CONFIG_COMPAT
+       .compat_ioctl = compat_ashmem_ioctl,
+#endif
```

Adding New Architectures

- In the root Android makefile config we added AArch64 as a new architecture
  - Allowed selecting the right toolchain and options
  - This triggers the right behaviour in almost all projects
  - The 'right behaviour' sometimes means refusing to build because support is missing
  - Projects with proper __LP64 support built 'out of the box'

- For Dalvik adding AArch64 beside AArch32 is not hard
  - Produce new config files and directories following the existing pattern

  - First build a VM using portable 'C' interpreter and no JIT
  - Implement, bytecode by bytecode, a fast interpreter
  - Build up the tracing JIT support the same way

  - Using VIXL from the JavaScript project was a significant help

Medium Task: Pixelflinger

- Includes ARM JIT for SW rendering
  - Very simple domain specific JIT
- Potentially optional
  - Ported to improve model performance
  - Useful experience
- Changes (Effort: ~6 weeks)
  - Enhanced IR to support distinct 'address' operations
  - Mapped to existing 32-bit operations on 32-bit targets
  - Addition of a new AArch64 Assembler and Disassembler
  - Uses CSEL for conditional execution
  - Pixelflinger uses a limited set of instructions

Large Task1: Bionic

- Effort is result of significantly cleaner AArch64 kernel code
  - Bionic completes the set of major libc implementations being ported to Aarch64
- AArch64 Linux kernel presents significantly simplified set of syscalls to world
  - Bionic port provides c wrappers to fill out full set of libc calls
  - Significant one-time effort was required to port and test
  - Porting was primarily a case of understanding AArch64 libc, AAPCS and kernel changes

Large Task 2: JavaScript

- Port of Google V8 JavaScript Engine is a significant effort
  - Large and rapidly evolving JIT
- Existing JIT supports several Architectures
  - Initial porting effort was to produce a compatible Assembler/Disassembler/Simulator library
  - Known as 'VIXL' this library is reusable for other ports
  - VIXL is already available under an open source license: https://github.com/armvixl/vixl
- Next effort was initial JIT
  - This is complete - we can run JavaScript on AArch64
- Final stage is porting optimising JIT
  - Underway

Large Task 3: Dalvik Porting

- The heart of Android
- 64-bit Java has been around since 2001 - It has been done before
- Three aspects to enabling Dalvik for 64-bit as well as the current 32-bit

- Core VM
  - Garbage Collection and object lifecycle
  - Execution of bytecodes via intepreter and JIT
  - Porting of Dalvik and Java APIs

- Objects and References
  - Support 64-bit addressing while mitigating increase in pointer size
- Implement a new JIT and Interpreter
- Fix APIs to use the right types at the right times

VM Changes and References

- Objects in Dalvik are 'reference heavy'
  - Risk that moving to 64-bit references will increase cache and memory pressure
- Solution is to use 'Compressed References' - common approach in 64-bit VMs
  - Objects references are 64-bit when handled by Interpreter and JIT (on-stack)
  - 32-bit when stored on the heap
  - Support for shifted references offers a possible heap size of 32GB (with 8-byte aligned objects)
  - Easy to move to 64-bit references when more than 32GB is needed
- Requires extensive VM changes
  - Has been working since March - passing Dalvik tests and is easily robust enough to boot Android
- Most changes are wrapping object references with 'expand/compress' functions

Longs, Ints, Pointers and Native code

- Changes are required where C pointers are used as handles
  - Java APIs that need to access native resources
  - Handle is typically pointer
- Typical model is to manage handle as primitive (historically in Dalvik as an int)
  - Needs to be changed to a long (64-bit value in Java) for 64-bit compatibility
  - Change is backwards compatible with 32-bit
- Many instances of a simple int to long change in API and native implementation
  - Google have patched these already in public tree
- In your own native code check for JNI code that casts pointers to or from jint
  - `return (jint)my_ptr;`

Fast Interpreter

- Optimised Assembler
  - But AArch64 programming model is similar enough to AArch32 for it to be straightforward
- In practice implementation is often simpler
- Same addressing modes for w, x, s and d registers
  - Allows uniform implementations for int, long, float, double operations
  - Support for float and double requires no fallback to libc
- Removal of conditional execution / addition of CSEL require some adjustments

```
cmp     r2, r3      ;; compare (vA, vB)
movne   r1, #2      ;; branch distance for not-taken
adds    r2, r1, r1 ;; convert to bytes, check sign
```

```
cmp   x2, x3          ;; compare (vA, vB)
mov   x0, #2          ;; x0<- not-taken in code units
csel  x1, x0, x1, ne ;; x1<-x0 if true (rev. of if)
adds  x2, x1, x1      ;; convert to bytes, check sign
```

JIT Implementations

- Work on a port of the Tracing JIT is underway
  - We are using VIXL to accelerate development
- Process is a matter of working through, bytecode by bytecode
  - New ISA makes for an easier implementation
  - The additional registers are very useful
  - Similar load/store addressing for all data makes things simpler

Summary on Dalvik

- Comments on progress and expectations
  - Dalvik applications will just run - the Java language is agnostic about this aspect of the hardware
  - Native code should be able to take advantage of the increased register size and number
  - We have a demo of 32-bit Android/64-bit kernel running on a Cortex A57 and Mali T621 FPGA
  - We also have a demo of current 64-bit Android progress running on an AEM Base Model
  - 32-bit Android/64-bit kernel is ready for productization now
  - 64-bit Android/64-bit kernel will be ready when it is needed

Conclusions

- Android has a Unix heritage
- Built on lots of open source, usually Linux, projects
  - These build for 64-bit already
- Most other general purpose code just needs minor changes
- Certain types of code require more work
  - VMs, JITs etc.
- Porting is a one-off, and costs can be shared across projects
  - Tools such as VIXL for example https://github.com/armvixl/vixl

Conclusions

- Porting C code to 64-bit is easy, porting to AArch64 is the same as any other platform

- AArch64 is a good target for code
- Offers a chance to reconsider design choices
  - Capabilities of new instruction set
  - Evolving form factors and models of use

- Use of models, adapting to new circumstances
  - Differentiation options
  - Time to market advantages
  - Opportunities to disrupt the market?

Thank You