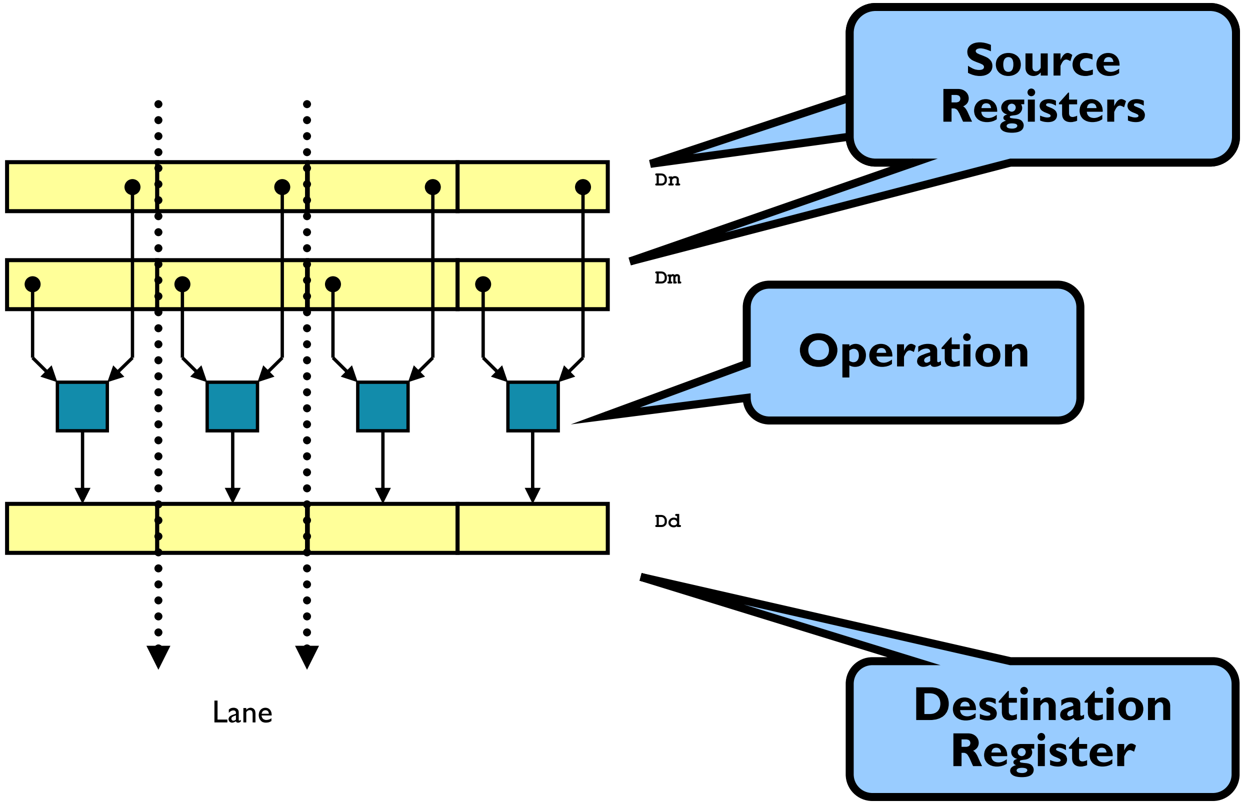
Moving your NEON optimizations to a 64-bit world

Ian Rickards
November 2014

What is NEON?

- NEON™ is a wide SIMD data processing architecture
  - Extension of the ARM® instruction set
  - 32 registers, 64-bit wide (**AArch64: 32 registers**, 128-bit wide)
- NEON Instructions perform “Packed SIMD” processing
  - Registers are considered as vectors of elements of the same data type
  - Data types can be: signed/unsigned 8-bit, 16-bit, 32-bit, 64-bit, single precision float (**AArch64: Double precision float**)
  - Instructions perform the same operation in all lanes



Source Registers

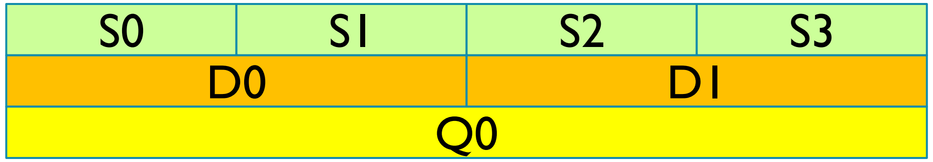
Operation

Destination Register

Changes in AArch64

- More registers
  - AArch32: 16x128-bit "Q-regs" Q0-Q15
  - AArch64: 32x128-bit "V-regs" V0-V31
- 'Dual view' no longer packed
  - 32 registers of each type: S, D, V
  - Clearer mapping of overlap
- Asm language changes
  - No 'v' in mnemonics
  - Width specifier moved to register description
  - 128-bit Q-regs renamed V-regs

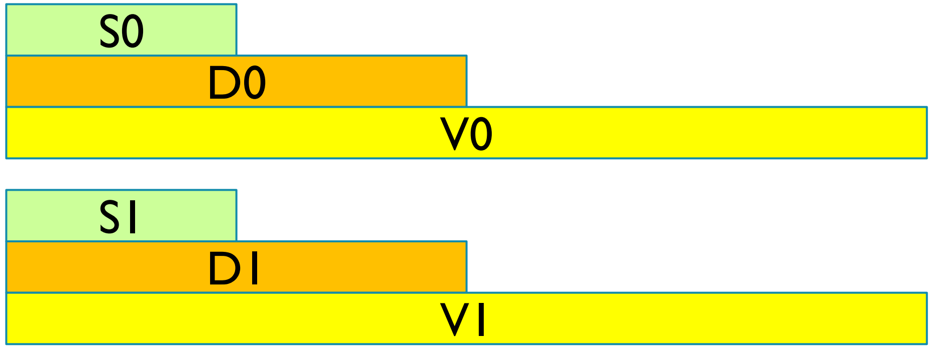
AArch32



vaddq.8 q0, q1, q2

8 = byte elements

AArch64



add v0.8B, v1.8B, v2.8B

8 = Number of elements

B	byte
H	halfword (16b)
S	single (32b)
D	double (64b)

New capabilities in AArch64 ‘AdvancedSIMD’

- Reduction across all lanes
  ADDV
- Rounding mode specified in instr
  - Not just ‘round to nearest’
- Double precision
- Single destination register
  VZIP, VSWAP now 2 instructions

- Crypto (ARMv8 but part of NEON)
- Ins instruction
- Table lookup larger table
- Saturating accumulate signed/unsigned
- AdvancedSIMD scalar
- REMOVED: high reg mapping, conditional execution

NEON use cases

- General purpose SIMD/DSP processing useful for many applications
- Support all new multimedia codecs

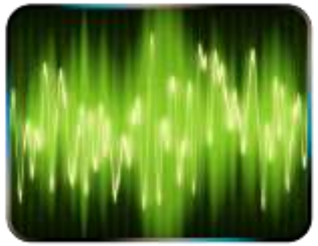
Watch any video in any format

Game processing

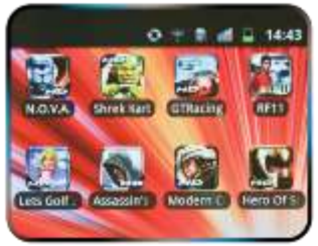
Edit & Enhance captured videos
Video stabilization

Process megapixel photos quickly

Antialiased rendering & compositing



Voice recognition



Advanced
User Interfaces

Powerful multichannel hi-fi
audio processing

NEON advantages

- **Easy to program**
  - Clean vector architecture
  - Off the shelf tools, OS support, commercial & opensource ecosystem support
- **Easy to debug**
  - Single flow of control
  - No separate DSP debugger

- **Fewer cycles needed**
  - Neon will provide real-world 1.5x - 4x performance on typical video codecs
  - Individual simple DSP algorithms can show larger performance boost (4x-8x)
  - Provides overall **power saving** and **increased processing capabilities**
  - **No overheads to ‘calling’ NEON**

How to use NEON

Automatically via OS

Opensource libraries

Vectorizing Compilers

- Uses NEON automatically from “C”

JIT compilers

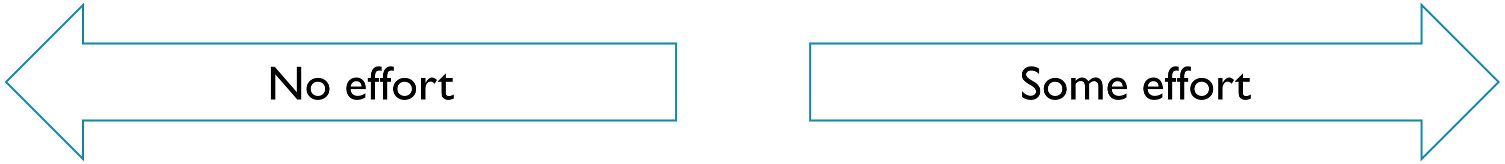
- LLVM (e.g. Android Renderscript)

Commercial vendors

- e.g. commercial HEVC decoder

C Instrinsics

Assembler


No effort

Some effort

Automatically via OS - NEON in Android

- Wide use of NEON optimizations in current Android source tree

- Many apps use NEON
  - Games (every game engine)
  - VR
  - Media editing / photo effects
  - Content creation

Component	ARMv7 NEON
VP8 (Google webm) decoder & encoder	YES (asm)
VP9 (Google webm) decoder & encoder	YES (asm)
JPEG	YES (asm)
Google WebP	YES (asm)
PNG decoder	YES (asm)
H.264 s/w decoder	YES (asm)
AMR WB encoder	YES (asm)
Skia	YES (asm)
WebRTC	YES (asm – FFT etc)
Renderscript	YES (via LLVM backend)
Blink (Chromium browser)	YES (intrinsics)

NEON optimizations in opensource

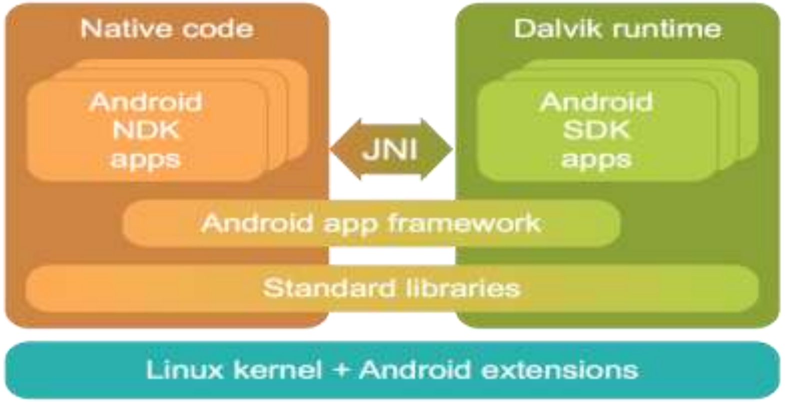
- **Google WebM** – **17,000** lines NEON code – both VP9 and VP8
- **Bluez** – official Linux Bluetooth protocol stack
- **Pixman** (part of cairo 2D graphics library)
- **ffmpeg (libav)** – **libavcodec -** LGPL media player
- **X264 -** GPL H.264 encoder – can be used for video conferencing
- **Eigen2** – C++ vector math / linear algebra template library
- **Theorarm** – libtheora NEON version (optimized by Google)
- **Android libjpeg / libjpeg-turbo** – optimized JPEG decode
- **libpng** – NEON optimized PNG decode
- **FFTW** – NEON enabled FFT library
- **Liboil / liborc** – runtime compiler for SIMD processing
- **webkit -** used by Google Chrome browser
- **Ne10** – library of low-level optimized math routines
- **Cocos2d-x** – 2D game engine uses Ne10
- **Skia** – 2D graphics library
- **Android Renderscript**

Android Native Development Kit (NDK) for ARM

- NDK is a toolkit to enable application developers to write native applications for the ARM processor

- NEON fully supported since NDK r5
- **64-bit support released in NDK r10 for "L"**

Android ABI	NEON support?
armeabi	No
armeabi-v7a	Optional - check cpu flags for NEON and ARMv8 crypto
arm64-v8a	**Yes: NEON always present**



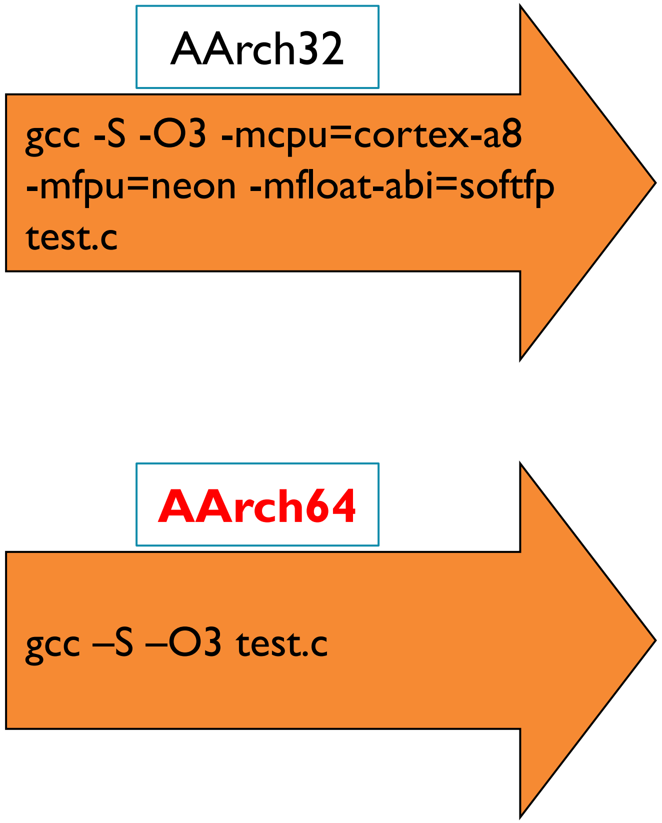
Android™ applications can be written in Java, native ARM code, or a combination of the two

Using vectorizing compiler – gcc

```
int a[256], b[256], c[256];

foo () {
  int i;

  for (i=0; i<256; i++){
    a[i] = b[i] + c[i];
  }
}
```



AArch32

```
gcc -S -O3 -mcpu=cortex-a8
-mfpu=neon -mfloat-abi=softfp
test.c
```

AArch64

```
gcc –S –O3 test.c
```

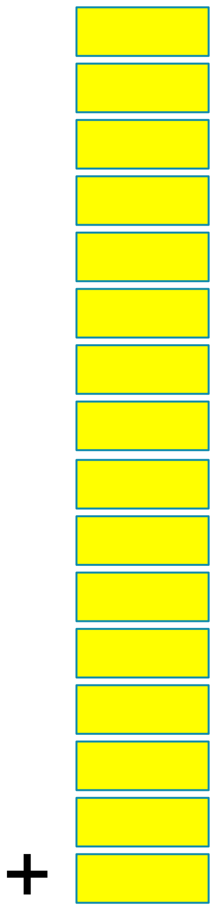
```
.L2:
        vldmia      r1!, {d18-d19}
        vldmia      r2!, {d16-d17}
        vadd.i32    q8, q9, q8
        vstmia      r3!, {d16-d17}
        cmp         r3, r0
        bne         .L2
```

```
.L2
        ldr     q0, [x0],16
        ldr     q1, [x2],16
        cmp     x0, x3
        add     v0.4s, v0.4s, v1.4s
        str     q0, [x1],16
        bne     .L2
```

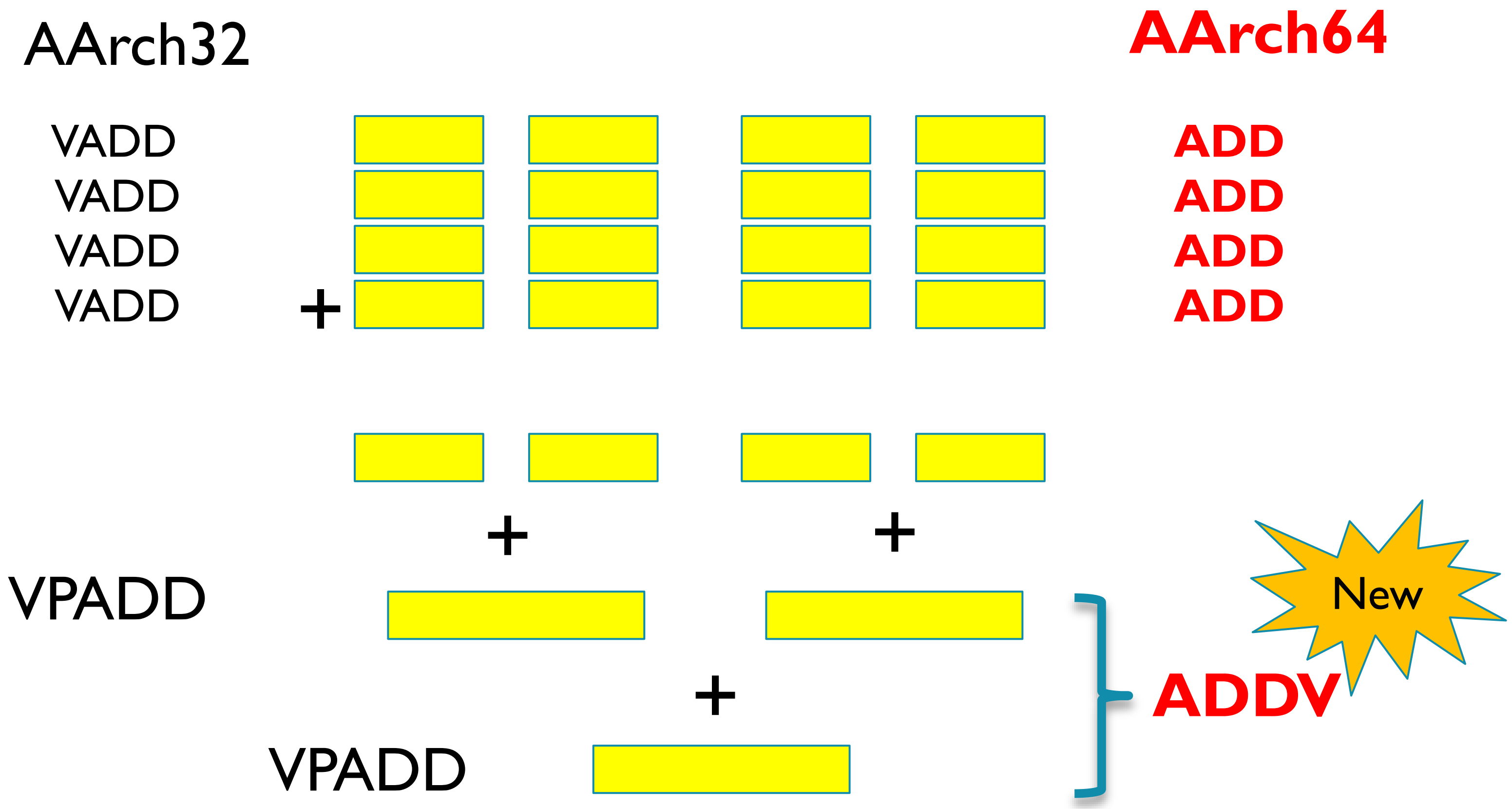
gcc -ftree-vectorize is default at –O3
example built with linaro-gcc-4.9-2014.05 aarch64 release

What is vectorizing?

- Adding 16x 32-bit values (scalar)



- Using NEON using Q-register



AArch32

VADD
VADD
VADD
VADD

VPADD

VPADD

AArch64

ADD
ADD
ADD
ADD

ADDV

New

Intrinsics

Fully compatible with AArch64

- **Include intrinsics header file (ACLE standard)**

  #include <arm_neon.h>

- **Use special NEON data types which correspond to D and Q registers, e.g.**

int8x8_t	D-register 8x 8-bit values
int16x4_t	D-register 4x 16-bit values
int32x4_t	Q-register 4x 32-bit values

- **Use NEON intrinsics versions of instructions**

```
vin1 = vld1q_s32(ptr);
vout = vaddq_s32(vin1, vin2);
vst1q_s32(vout, ptr);
```

- **Strongly typed!**
  - Use vreinterpret_s16_s32( ) to change the type

```
static inline void Filter_32_opaque_neon(unsigned x, unsigned y,
                                SkPMColor a00, SkPMColor a01,
                                SkPMColor a10, SkPMColor a11,
                                SkPMColor *dst) {
    uint8x8_t vy, vconst16_8, v16_y, vres;
    uint16x4_t vx, vconst16_16, v16_x, tmp;
    uint32x2_t va0, va1;
    uint16x8_t tmp1, tmp2;

    vy = vdup_n_u8(y);                // duplicate y into vy
    vconst16_8 = vmov_n_u8(16);       // set up constant in vcon
    v16_y = vsub_u8(vconst16_8, vy);  // v16_y = 16-y

    va0 = vdup_n_u32(a00);            // duplicate a00
    va1 = vdup_n_u32(a10);            // duplicate a10
    va0 = vset_lane_u32(a01, va0, 1); // set top to a01
    va1 = vset_lane_u32(a11, va1, 1); // set top to a11

    tmp1 = vmull_u8(vreinterpret_u8_u32(va0), v16_y); // tmp1 =
    tmp2 = vmull_u8(vreinterpret_u8_u32(va1), vy);    // tmp2 =
```

NEON intrinsics

Pros

- Readability
- Reusability (inline functions, templates)
- Type checking (vreinterpret)
- Easier to debug
- **Portability to AArch64**
- Compiler can combine instructions (e.g. MAC)
- Compiler does register allocation
- Compiler does instruction scheduling

Cons

- Little control over registers used
- Does not always generate the code you expect

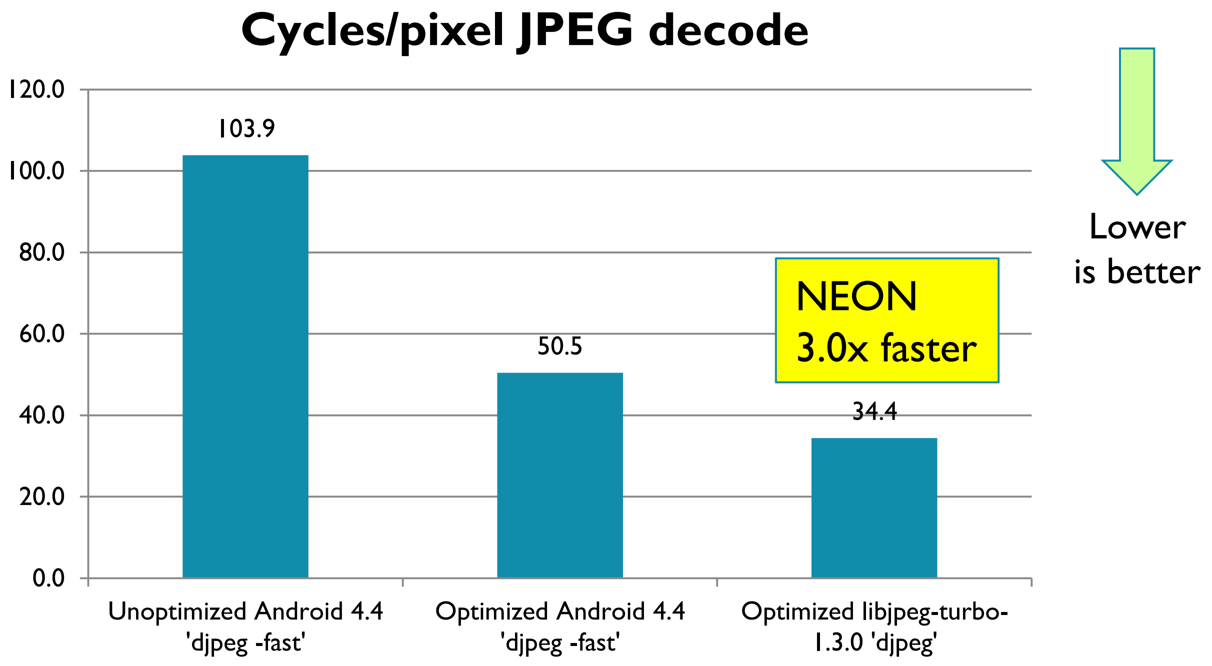
Compatibility

- C/instrinsics will port with no effort
- Asm requires reworking of .s file (mostly cosmetic, but can take advantage of additional registers)
- AArch64 NEON optimization in progress
  - ARM & Linaro working on key Android libraries using intrinsics
  - ffmpeg AArch64 NEON decoders (asm)
  - X264 AArch64 NEON encoder (asm)

AArch64 NEON coding technique	**Compatible?**
Vectorized "C"	Fully compatible
Intrinsics ("arm_neon.h")	**Fully compatible**
Asm (.s)	Some porting required
Library routines	Yes, if library available

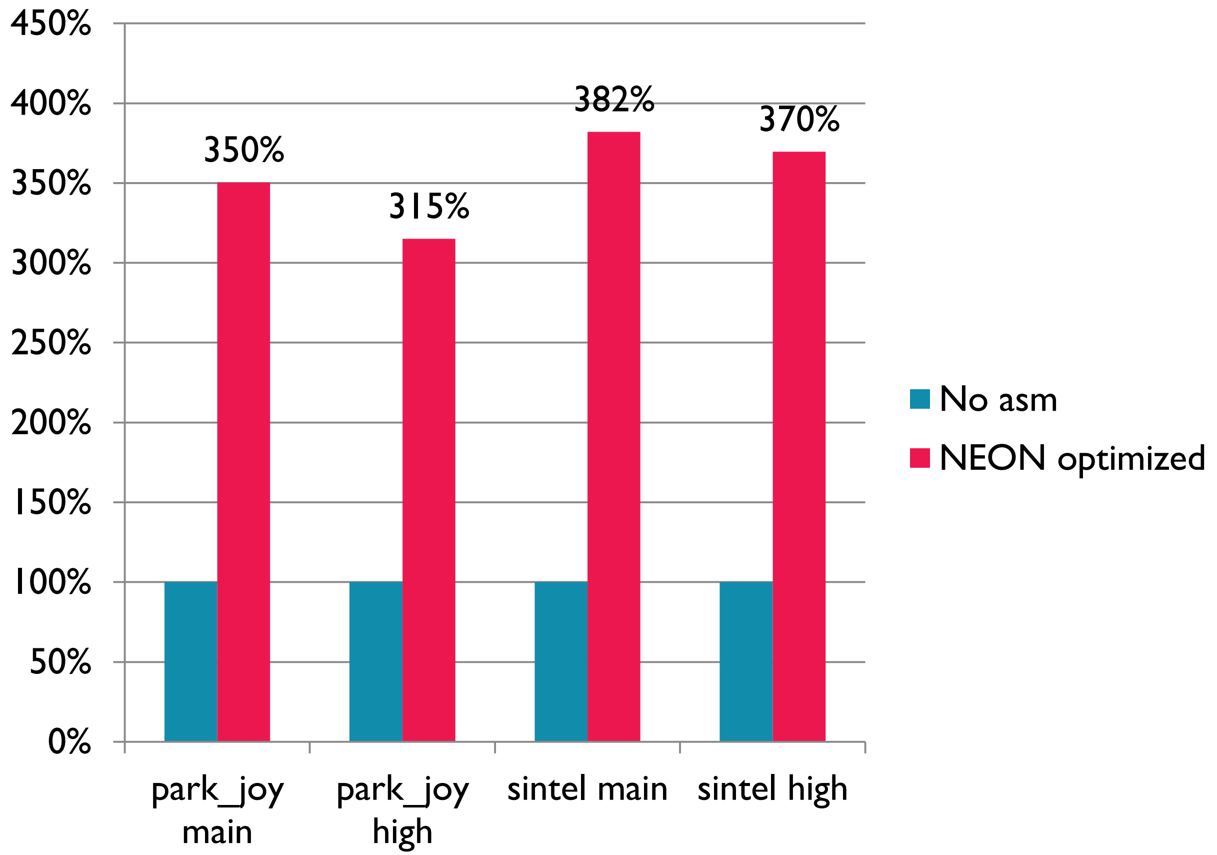
Fastest JPEG codecs: Android & libjpeg-turbo

- NEON optimizations integrated into
  - Official Android 4.4 (Kitkat) and later (plus partner-specific versions)
  - Libjpeg-turbo (opensource)
- Significantly improves speed of multi-megapixel image decode
- Benchmarked on 1.7GHz ARM Cortex®-A15
  - Optimized 34.4cycles/pix => 0.4s total for image
- Test image: 19.4Mpix
  http://commons.wikimedia.org/wiki/File:Willaerts_Adam_The_Embarkation_of_the_Elector_Palantine_Oil_Canvas-huge.jpg



X264 – high quality H.264 encoding

- Can be used for highest-quality offline encoding – movies & tv content for on-demand services
- Full ARMv7 NEON optimizations
  - 5300 lines of NEON asm
- **New**: AArch64 NEON (Aug 2014)
- Performance results from Cortex-A15 processor @1.7GHz



File (media.xiph.org)	
park_joy_420_720p50.y4m	1280x720
sintel_trailer_2k_480p24.y4m	854x480

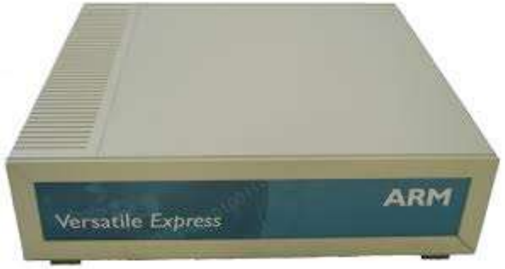
Wide range of NEON enabled low-cost dev boards

- Odroid XU3 $179
  - 4x 2.0GHz Cortex-A15 ‘Octa’ b.L

- Cubieboard4 CC-A80
  - 4x 2.0GHz Cortex-A15 + 4x 1.3GHz Cortex-A7

- Cubietruck $65
  - 4xCortex-A7

- Chromebook2
  - 4xCortex-A15 ‘Octa’ b.L



- **64-bit: ARM “Juno”**

- **64-bit: Nexus 9**

NEON summary

- NEON in AArch64 is much improved
  - More registers
  - New instructions
  - Cleaner instruction set

- Migrating to 64-bit
  - Use C or NEON intrinsics for best portability
  - Asm best in special circumstances, e.g. video codecs

    Normally straightforward to port ARMv7 NEON to AArch64 NEON
  - NDK r10 provides full support – **start testing apps now!**

- Existing NEON documentation still very relevant
  - NEON Programmer's Guide
  - Blog entries
  - http://www.arm.com/community/

- Tune for AArch64
  - Extra registers
  - Double precision float
  - New instructions

Thank You

The trademarks featured in this presentation are registered and/or unregistered trademarks of ARM Limited (or its subsidiaries) in the EU and/or elsewhere. All rights reserved. Any other marks featured may be trademarks of their respective owners