ARM NEON™ Instruction Set and Why You Should Care

The future's so bright, I gotta wear shades?

Mike Anderson
Chief Scientist
The PTR Group, Inc.
http://www.theptrgroup.com

Embedded Linux Conference
April 11 - 13, 2011
Hotel Kabuki
San Francisco, CA

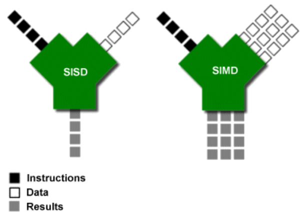
What We Will Talk About

- Parallelism in computing
- The evolution of ARM SIMD instructions
- What is NEON™?
- NEON™ Architecture Overview
- Developing Code for NEON™
- Example Performance Improvements
- Summary

Source: SmashingMagazine.com

Parallelism In Computing

- The ability to execute operations in parallel allows us to achieve more than simple clock speed improvements
- Flynn's Taxonomy describes various flavors of parallelism
  - SISD: Single Instruction/Single Data
    - Traditional scalar processors
  - SIMD: Single Instruction/Multiple Data
    - Vector processors (data parallelism)
  - MIMD: Multiple Instruction/Multiple Data
    - Multi-core processors (thread parallelism)



Source: ARS Technica

Data Parallelism

- SIMD computing is well suited to a wide variety of algorithms
  - FIR, FFT, dot product, multiply/accumulate (MAC), color conversion and more
- The idea is to load up multiple pieces of data and perform an operation across all of the data at once
  - Examples include x86 MMX/SSE, XScale WMMX, and ARM NEON™
- SIMD operation requires some thought to be able to "vectorize" the code

Evolution of ARM Instructions

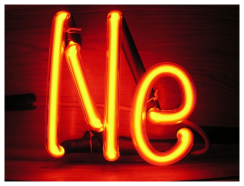
- The ARMv5TE introduced "enhanced DSP extensions"
  - 2-3x DSP performance boost over entry ARMv5
    - Load and store instructions for pairs of registers with new addressing modes
    - Single-cycle 32x16 and 16x16 multiply/accumulate (MAC) features
    - CLZ instruction to boost divide and normalization speed
    - Saturation extension to existing instructions
- Targeted at applications such as audio CODECs, networking, etc. that required a mix of DSP and control

The ARM Architecture and SIMD

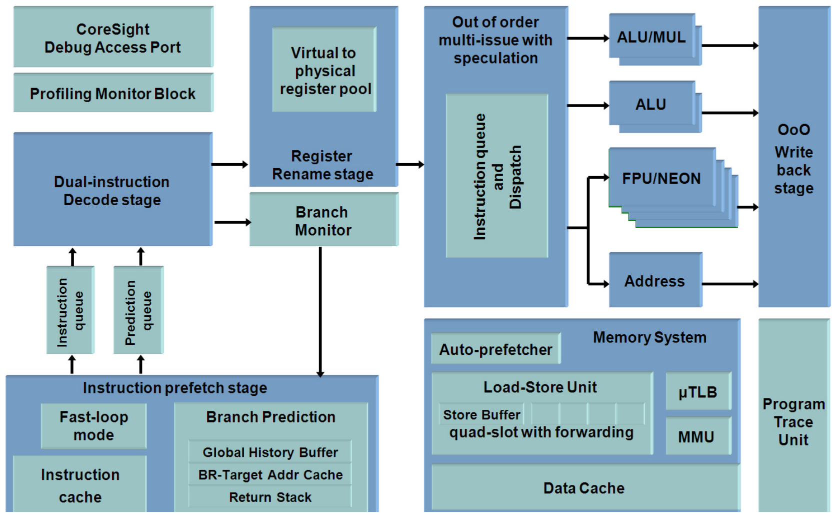
- The SIMD capabilities were first introduced to ARM with the ARMv6 architecture
  - ARM11 had a simple SIMD unit in coordination with the VFPv2 floating-point unit
- 60+ new SIMD instructions
  - 32x32 fractional MAC
  - Concurrent 8/16-bit operations
  - Concurrent computation of 2x16-bit or 4x8-bit operands
  - Dual 16x16 multiply-add/subtract
- 2x ARMv5TE performance for media applications

ARMv7 and NEON™

- The NEON™ unit is billed as a "media engine" within the ARM core
  - Separate register file and 10-stage execution pipeline from ARM core
  - Optional as is the VFPv3 floating point unit
  - Targets media acceleration, signal processing and graphics
- Over 100 SIMD instruction types available in both ARM and Thumb-2 mode
  - Instructions run intermixed with standard ARM instructions
  - Supports integer and floating point operations


Source: geocaching.com

ARM Cortex A9



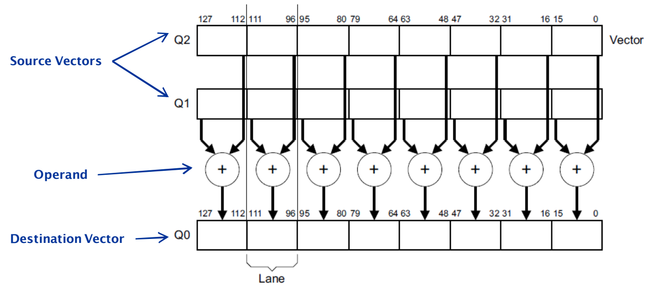
Source: ARM

What is NEON™?

- Neon™ is a "packed SIMD" processing unit
  - 32, 64-bit wide registers
    - Dual-view as 16, 128-bit registers
  - Registers are treated as vectors of elements of the same data type
- Supports both signed and unsigned 8, 16, 32 and 64-bit integers and 32-bit IEEE 754 single precision floats
- Instructions perform the same operation in all lanes

Operand Flow

- 1-2 source registers and a destination register
  - Allows for mixing of double and quad-word operands and destinations



Source: ARM

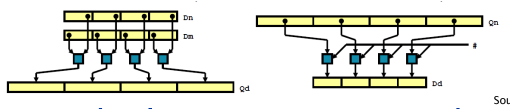
VADD.I16 Q0, Q1, Q2

NEON™ Operations

- SIMD instructions supported by vectorizing compiler, intrinsic functions or in-line assembly
- Instructions include:
  - 1,2,4-byte load/store across vectors
  - Add/subtract with saturation, halving and rounding
    - MIN, MAX, NEG, MOV, ABS, ABD
  - Multiplication (MUL, MLA, MLS and more)
  - Shifts with saturation, rounding
  - Compares and selection
  - Logical operators
  - Bit fields, shuffles (ZIP, UZIP)
  - Reciprocal estimate/step, Square root estimates
  - And many more…

Promotion/Demotion

- NEON™ can use both register views in the same instruction
  - Allows promotion/demotion of elements within a single operation

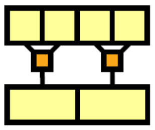


Source: ARM

- Operations can be long, narrow or wide
  - Long operations promote elements to double the precision
    - E.g., VMULL, VADDL 16x16->32
  - Narrow operations demote like narrowing
    - E.g., VADDHN
  - Wide operations promote the 2nd operand (16+32-> 32)
    - E.g., VADD, VSUB

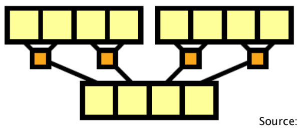
Pair-wise Operations

- NEON™ supports pair-wise operations that work on adjacent registers and perform operations like reductions
  - ADD, MIN, MAX
- Long



Source: ARM

- Normal



Source: ARM

Load/Store Operations

- Many different memory access patterns can be achieved with a single instruction
- Understands structure loads and unaligned transfers
  - Minimizes the need for padding typically found in structures
- Can load from/store to RAM, registers, immediate values

Developing Code for NEON™ #1

- There are several ways to develop code that uses the NEON™ unit
- Use a vectorizing compiler
  - The compiler will look for code that can be vectorized automatically
    - Success requires you to code your algorithms to help the compiler vectorize
      - E.g., Force operations in groups of four

```
void add_ints(int * __restrict pa, int * __restrict pb, unsigned
   int n, int x)
{
   unsigned int i;
   for(i = 0; i < (n & ~3); i++)
     pa[i] = pb[i] + x;
}
```

- To enable vectorization you must add -mfpu=neon -ftree-vectorize to gcc
  - `arm-none-linux-gnueabi-gcc -mfpu=neon -ftree-vectorize -c vectorized.c`

Developing Code for NEON™ #2

- Another approach is to use the NEON™ intrinsic operators
  - Translates to NEON™ assembler operations

```
#include <arm_neon.h>
uint32x4_t double_elements(uint32x4_t input)
{
   return(vaddq_u32(input, input));
}
```

- To compile with gcc:
  - `arm-none-linux-gnueabi-gcc -mfpu=neon intrinsic.c`
- A complete list of supported intrinsics can be found at:
  - http://gcc.gnu.org/onlinedocs/gcc/ARM-NEON-Intrinsics.html#ARM-NEON-Intrinsics

Developing Code for NEON™ #3

- If you're really hard-core, you can write in assembly:

```
   .text
   .arm
   .global double_elements
double_elements:
   vadd.i32 q0,q0,q0
   bx lr
   .end
```

- Then, assemble with gas:
  - `arm-none-linux-gnueabi-as -mfpu=neon asm.s`
- Or, take the way of the weak and use an optimized library

Source: Paramount

Neon™ Optimized Libraries

- The OpenMAX DL library from the Khronos Group is a NEON™ optimized, royalty-free library for AV CODEC acceleration
  - http://www.khronos.org/openmax/
- Supports:
  - MPEG-4 and H.264 baseline
  - JPEG, Colorspace conversion, rotates, scaling, composting
  - MP3, AAC
  - FIR, IIR, FFT, Dot Product
- Code samples and library available from ARM site (https://silver.arm.com/browse/OX000)
  - Requires registration and login

Relative Performance Metrics

- The use of NEON™ shows 1.6-2.5x performance boost over ARM11 in complex video CODEC (MPEG4)
- Audio processing FFT (used in AAC, voice recognition, etc.)

ARM11 (v6 SIMD)	15.2us
ARM Cortex A8 (v7 NEON)	3.8us

  - Both were hand-tuned assembly
- FFT in ffmpeg is 12x faster on Cortex-A8 in hand-tuned asm

Open Source with NEON™

- There are several OSS projects that are NEON™ enabled including:
  - BlueZ Linux Bluetooth stack
    - NEON™ sbc audio encoder
  - ffmpeg – libavcodec
    - NEON™ video: MPEG-2, MPEG-4 ASP, H.264 (AVC), VC-1, VP3, Theora
    - NEON™ audio: AAC, Vorbis, WMA
  - Android Skia library
    - S32A_D565_Opaque is 5x faster w/ NEON™
  - Eigen2 C++ vector math linear algebra library
  - x264 – GPL H.264 encoder for video conferencing
  - Pixman (part of cairo 2D graphics library)
    - Compositing/alpha blending 8x faster w/ NEON™

Power Efficiency of NEON™

- NEON™ unit is approximately the same mW/MHz as integer core
  - Power savings from lower MHz and/or removing special-purpose media accelerators
- Example YouTube HQ video (480x270 H.264 @ 30fps)
  - ffmpeg player on 500MHz Cortex-A8 decoding to YUV420
  - 64fps video only
    - 234MHz average
  - 60fps including AAC-LC audio
    - 250MHz average
- Beagle C4 is roughly 1mW/MHz including memory accesses

Summary

- For the right application, the NEON™ unit can yield 2-8x performance boost
- Using pre-optimized libraries reduces the development headaches while yielding good results
  - Strong 3rd party ecosystem of library vendors
- Hand-coding assembler or using NEON™ intrinsics requires experimentation to get the instruction mix right
  - Always dump assembly output and examine