



BUD17-117

# Auto Vectorization Support in OpenJDK 9 AArch64 Hotspot C2 Compiler

Yang Zhang  
Stuart Monteith





Linaro  
connect

Budapest 2017

ENGINEERS  
AND DEVICES  
WORKING  
TOGETHER

# Overview

- History
- Current NEON support in AArch64 backend
- How to add a NEON instruction
- Potential enhancements
  - AArch64 specific
  - Architecture independent



Linaro  
connect

Budapest 2017

ENGINEERS  
AND DEVICES  
WORKING  
TOGETHER

# Overview

- History
- Current NEON support in AArch64 backend
- How to add a NEON instruction
- Potential enhancements
  - AArch64 specific
  - Architecture independent

# History

- Linaro has been involved into OpenJDK aarch64 support since 2014. Edward Nevill has given OpenJDK updates in previous Connects.
  - Linaro OpenJDK status: <http://openjdk.linaro.org/sfo15-openjdk.pdf>
  - OpenJDK introduction: <http://openjdk.linaro.org/lca15/OpenJDK.pdf>
- Vectorization support
  - On x86, vectorization optimizations have been added to hotspot-server since 2012.
    - 6340864: Implement vectorization optimizations in hotspot-server
  - On AArch64, thanks to Ed's work, most of NEON instructions have been added.
    - 8086087: aarch64: add support for 64 bit vectors
    - 8079565: aarch64: Add vectorization support for aarch64





Linaro  
connect

Budapest 2017

ENGINEERS  
AND DEVICES  
WORKING  
TOGETHER

# Overview

- History
- Current NEON support in AArch64 backend
- How to add a NEON instruction
- Potential enhancements
  - AArch64 specific
  - Architecture independent

# Current NEON support in AArch64 backend

Operations	supported	unsupported
Load/store	ldr/str S/D/Q	ldn/stn (n= 1,2,3,4)
Replicate	movi B/S, dup H/S/D	
Reduction arithmetic	addw, addv, mul, fadds/d, fmul/d	
General arithmetic	add/fadd, sub/fsub, mul/fmul, fdiv, mla/mls, fsqrt, fabs	addhn/addhn2, addp/faddp, sadalp, saddl/saddl2, saddlp, saddlv, saddw/saddw2, hadd, rsubhn/rsubhn2, shsub, subl/subl2, subw/subw2, pmul, pmull/pmull2, smull/smull2, umull/umull2, dmulh, dmull/dmull2, fabd, sabs, sabal/sabal2, sabd, sabdl
Logical	fneg, and, or, eor	not, orn, rbit
Shift	sshl/ushl/shl, sshr/ushr	shll/shll2, shrn/shrn2, sli
Others		bic, bif, bit, bsl, cls, clz, cmeq/ge/gt/le/lt/tst, cnt, ext, rev16/32/64, xtn/xtn2, trn1/trn2, zip1/2, uzip1/2, s/u/fmax, s/u/fmaxp, s/u/fmaxv, s/u/fmin, s/u/fminp, s/u/fminv



Linaro  
connect

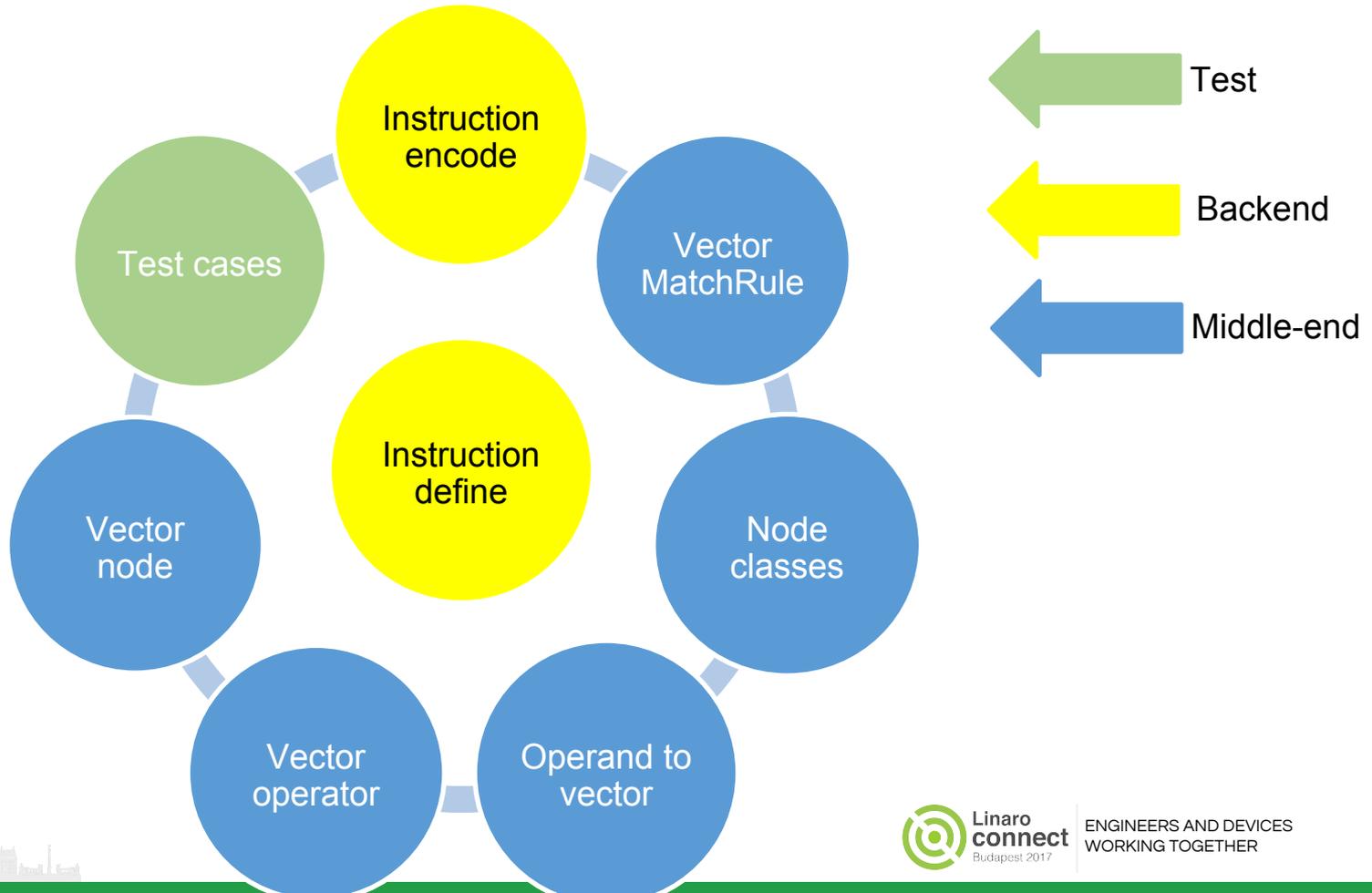
Budapest 2017

ENGINEERS  
AND DEVICES  
WORKING  
TOGETHER

# Overview

- History
- Current NEON support in AArch64 backend
- How to add a NEON instruction
- Potential enhancements
  - AArch64 specific
  - Architecture independent

# How to add a NEON instruction



# How to add a NEON instruction

```
An example in hotspot/src/cpu/aarch64/vm/aarch64.ad: // the Architecture Description File

instruct vadd4l(vecX dst, vecX src1, vecX src2) // vecX: 128-bit vector register
%{
  predicate(n->as_Vector()->length() == 4); // Predicate vector size
  match(Set dst (AddVI src1 src2)); // Match rule, AddVI is vector add int ideal node
  ins_cost(INSN_COST); // Basic cost
  format %{ "adv $dst,$src1,$src2\t# vector (4S)" %} // Format for C2 disassembly
  ins_encode %{ // Encode it in aarch64
    __ adv(as_FloatRegister($dst$$reg), __ T4S,
           as_FloatRegister($src1$$reg),
           as_FloatRegister($src2$$reg));
  %}
  ins_pipe(vdop128); // Pipeline is NEON 128-bit operation
%}
```





Linaro  
connect

Budapest 2017

ENGINEERS  
AND DEVICES  
WORKING  
TOGETHER

# Overview

- History
- Current NEON support in AArch64 backend
- How to add a NEON instruction
- Potential enhancements
  - AArch64 specific
  - Architecture independent

# AArch64 specific: vector MLA

```
public static int vectSumOfMulAdd(  
    int[] a,  
    int[] b,  
    int[] c,  
    int[] d) {  
    int total = 0;  
    for (int i = 0; i < LENGTH; i++) {  
        d[i] = (int)(a[i] * b[i] + c[i]);  
        total += d[i];  
    }  
    return total;  
}
```

Code snippet produced by C2:

```
ldr q18, [x19,#32]  
ldr q17, [x4,#32]  
ldr q19, [x20,#32]  
mul v17.4s, v17.4s, v18.4s  
add v17.4s, v17.4s, v19.4s
```

Should be optimized to:

```
ldr q18, [x19,#16]  
ldr q17, [x20,#16]  
ldr q16, [x4,#16]  
mla v18.4s, v16.4s, v17.4s
```



# AArch64 specific: vector MLA

- Issue

- The vector MLA instruction has been added to aarch64.ad, but it can't be generated for some simple test cases.

- Direct cause in AArch64 backend

- Add the new mla matching rules which swap output of MulVI and dst.
- Patch details: <http://cr.openjdk.java.net/~njian/8169697/webrev.00/>

- Root cause

- Add AddVB/S/I/L/F/D nodes to commut\_op\_list.
- Patch details: <http://cr.openjdk.java.net/~njian/8169697/webrev.share/>
- Upstream prefer the second patch. We missed jdk 9 with this one. It needs testing on all platforms. It might be merged into jdk 10.



# AArch64 specific: byte NEON instructions

```
public static void vectAddByte(  
    byte[] a,  
    byte[] b,  
    byte[] c) {  
    for (int i = 0; i < LENGTH; i++) {  
        c[i] = (byte)(a[i] + b[i]);  
    }  
}
```

Code snippet produced by C2:

```
ldrsw w10, [x6,w18,sxtw #0]  
ldrsw w12, [x3,w18,sxtw #0]  
add    w19, w12, w10  
strb   w19, [x4,w18,sxtw #0]
```



# AArch64 specific: byte NEON instructions

- Issue

- Vectorized instructions for byte data type are not generated

- Direct cause in AArch64 backend

- Add 32-bit vector register vecS to enable byte data vectorization.
- The solution can optimize example to:

```
ldr    s17, [x16,#16]
```

```
ldr    s16, [x15,#16]
```

```
add    v16.8b, v16.8b, v17.8b
```

```
str    s16, [x13,#16]
```

- The micro benchmark of vectAddByte shows about 3.7 times improvement.
- The patch is in progress because it's not robust enough.

- Root cause in middle-end

- Compiler doesn't use vector registers as wide as it could because there's no enough unrolling of the loop at the time vectorization happens. This issue will be discussed later.



# AArch64 specific: interleaving load/store

```
// DIM is 2 here
public static void vect2DAddInt(
    int[] out,
    int[] in,
    int[] c) {
    int c0 = c[0];
    int c1 = c[1];
    for (int i = 0; i < VECT_LENGTH; i++) {
        out[i*DIM] = in[i*DIM] + c0;
        out[i*DIM + 1] = in[i*DIM + 1] - c1;
    }
}
```

There is no NEON instruction generated by C2 compiler

Gcc can optimize similar C code to assembly:

```
ld2 {v2.4s, v3.4s}, [x10], #32
add v0.4s, v5.4s, v2.4s
sub v1.4s, v3.4s, v4.4s
st2 {v0.4s, v1.4s}, [x9], #32
```



# AArch64 specific: interleaving load/store

- Issue

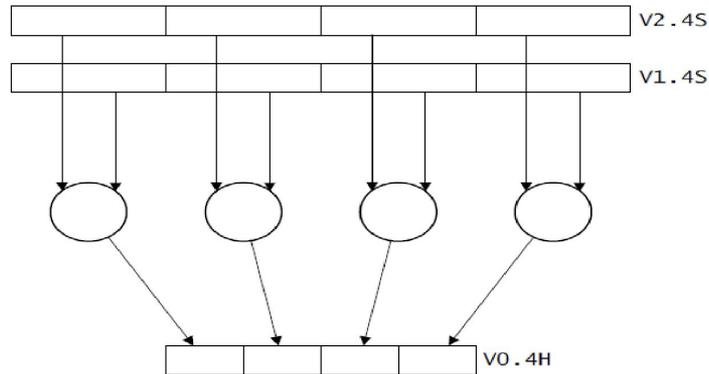
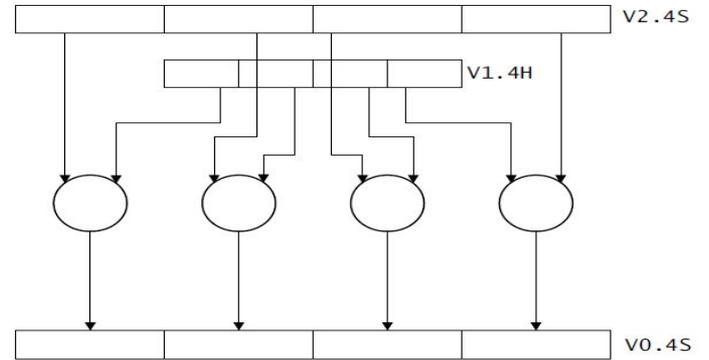
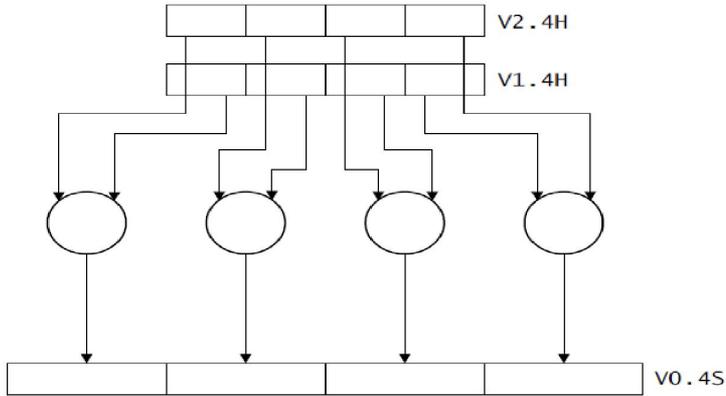
- Interleaving ldn/stn (n = 1,2,3,4) instructions are not supported.

- Possible solutions

- The work isn't started. But the solution might be separated into two steps:
- Add fully support for ld1/st1
  - Currently, ld1/st1 S/D/Q is supported in AArch64 backend. But in x86 backend, there are load/storeV32 and load/storeV64. So that ld1/st1 {v0, v1}/{v0, v1, v2, v3} might be added to AArch64 backend first.
- Add support for ldn/stn (n=2,3,4)
  - Such multiple N-element structures load/store instructions are ARM specific, lack of support from middle-end. How to add support in middle-end is a challenge.



# AArch64 specific: vectorized long/wide/narrow



# AArch64 specific: vectorized long/wide/narrow

```
public int SumSquareError(short[] a, short[] b) {  
    int sse = 0;  
    for (int i = 0; i < LENGTH; i++) {  
        int diff = a[i] - b[i];  
        sse += diff * diff;  
    }  
    return sse;  
}
```

There is no NEON instruction generated by C2 compiler

Gcc can optimize similar C code to assembly:

```
ldr q1, [x11,x7]  
ldr q3, [x3,x7]  
ssubl v2.4s, v1.4h, v3.4h  
ssubl2 v1.4s, v1.8h, v3.8h  
mla v0.4s, v2.4s, v2.4s  
mla v0.4s, v1.4s, v1.4s  
addv s0, v0.4s
```



# AArch64 specific: vectorized long/wide/narrow

- Issue

- Vectorized long/wide/narrow instructions are not supported.

- Possible solutions

- The work isn't started. To implement these instructions in AArch64 backend, that input/output or two inputs have different width is a challenge for vector node implementation in middle-end.



# Architecture independent: byte and short support

Java	<pre>public static void vectAddShort(     short[] a,     short[] b,     short[] c) {     for (int i = 0; i &lt; LENGTH; i++) {         c[i] = (short)(a[i] + b[i]);     } }</pre>
X86 C2	<pre>vmovq 0x10(%rbx,%r13,2),%xmm0 ----- vmovq accesses 64 bits memory vpaddw 0x10(%rcx,%r13,2),%xmm0,%xmm0 vmovq %xmm0,0x10(%rdx,%r13,2)</pre>
AArch64 C2	<pre>ldr d17, [x15,#16] ----- d register is 64 bits ldr d16, [x13,#16] add v16.4h, v16.4h, v17.4h str d16, [x10,#16]</pre>

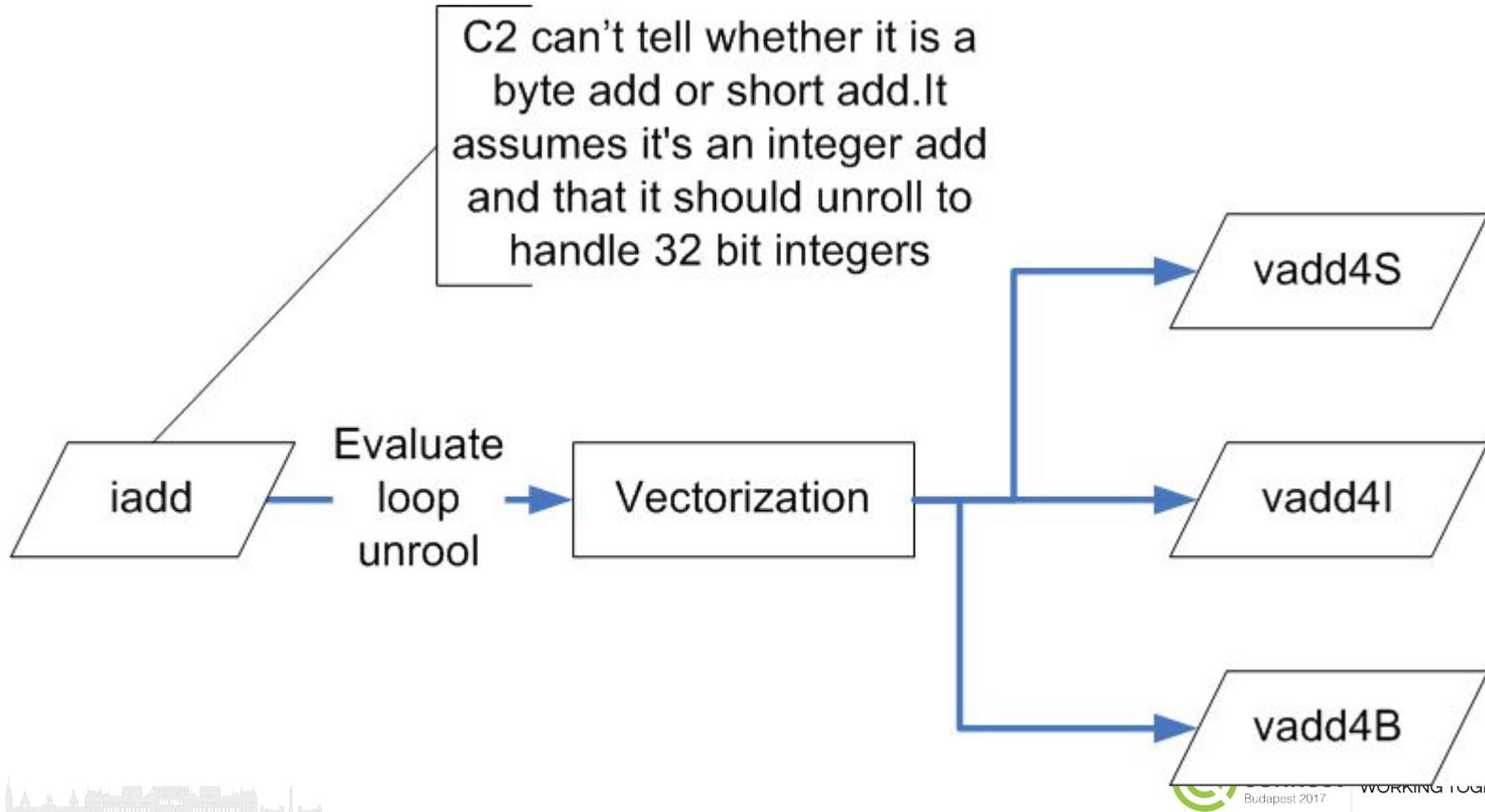


# Architecture independent: byte and short support

Java	<pre>public static void vectAddByte(     byte[] a,     byte[] b,     byte[] c) {     for (int i = 0; i &lt; LENGTH; i++) {         c[i] = (byte)(a[i] + b[i]);     } }</pre>
X86 C2	<pre>vmovd 0x10(%r9,%r14,1),%xmm2 ----- vmovd accesses 32 bits memory vpaddb 0x10(%r11,%r14,1),%xmm2,%xmm2 vmovd %xmm2,0x10(%rsi,%r14,1)</pre>
AArch64 C2	<pre>ldr s17, [x16,#16] ----- s register is 32 bits ldr s16, [x15,#16] add v16.8b, v16.8b, v17.8b str s16, [x13,#16]</pre>



# AArch64 specific: byte and short support





# Thank You

#BUD17

For further information: [www.linaro.org](http://www.linaro.org)

BUD17 keynotes and videos on: [connect.linaro.org](http://connect.linaro.org)

