Vector Codegen in the RISC-V Backend

Luke Lau

11th October 2023

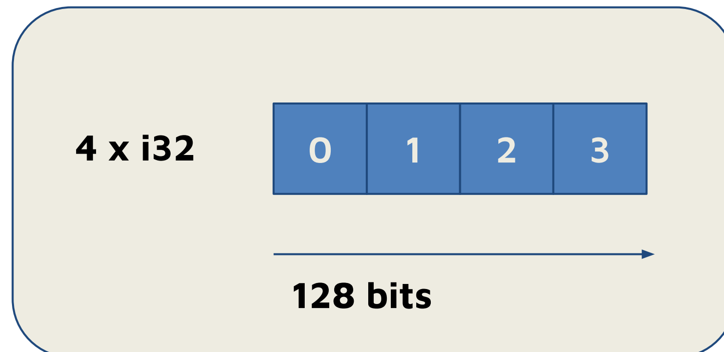
Agenda

- Overview of RISC-V Vector ISA
- Modelling semantics in LLVM IR
- Vector codegen and lowering
- Middle-end vectorization passes

Arm NEON

```
%v = add <4 x i32> %x, %y
```

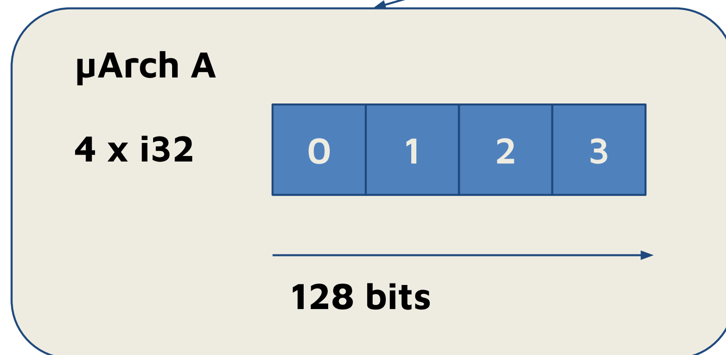
```
add v0.s, v1.s, v2.s
```



Arm SVE

- **Scalable** vector registers
  - 128-1024 bits in size (128-2048 in SVE2)
- Code is **vector length agnostic** (VLA)

```
add z0.s, z1.s, z2.s
```

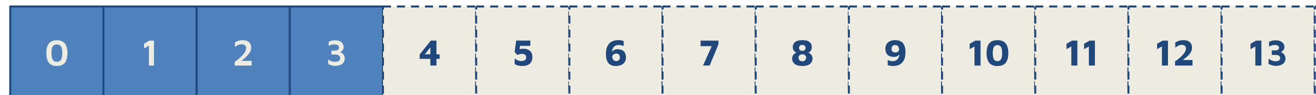


Arm SVE

```
%v = add <???> %x, %y
```

```
add z0.s, z1.s, z2.s
```

n x 4 x i32



Min 128 bits

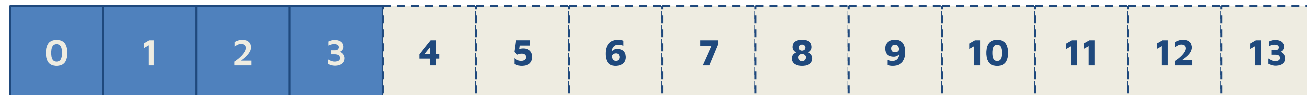
Scalable vectors

- **vscale**: constant **unknown at compile time**
- On SVE, **vscale = register size in bits / 128**

```
%v = add <vscale x 4 x i32> %x, %y
```

```
add z0.s, z1.s, z2.s
```

vscale x 4 x i32



Min 128 bits

RISC-V “V” Vector Extension (RVV)

- Standard RISC-V extension that provides vector capabilities
- Ratified in 2021
- Adds 32 **VLEN**-sized vector registers v0-v31
  - 32 <= **VLEN** <= 65,536
  - Minimum supported VLEN in LLVM is 64 bits
  - **vscale = VLEN / 64**

RISC-V “V” Vector Extension (RVV)

```
%v = add <vscale x 2 x i32> %x, %y
```

```
vsetvli t0, zero, e32, m1, ta, ma

vadd.vv v0, v1, v2            add z0.s, z1.s, z2.s
```

Where do we encode the element type?

- Element size **SEW** is encoded in **VTYPE** register
- Set **VTYPE** with **vsetvli/vsetivli/vsetvl**

Vector length VL

- Can control the number of elements to operate on with the **VL** register
- Configured with vset[i]vli

```
vsetivli t0, 5, e32, m1, ta, ma

vadd.vv v1, v2, v3
```

Mask

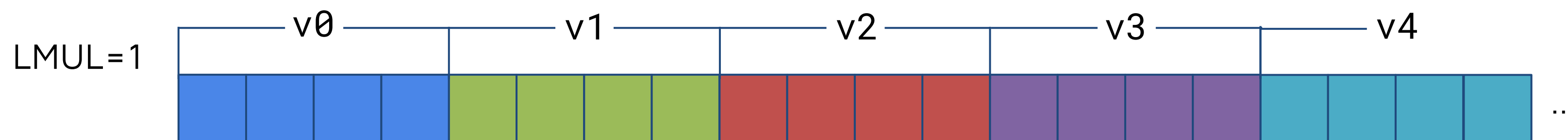
- Can predicate operations by using a mask register
- Always has to be in **v0**

```
vsetivli t0, 5, e32, m1, ta, ma
vadd.vv v1, v2, v3, v0.t
```

Register grouping LMUL

- Controlled by length multiplier **LMUL**, stored in **VTYPE**
- At LMUL=1, can operate on **VLEN / SEW** elements

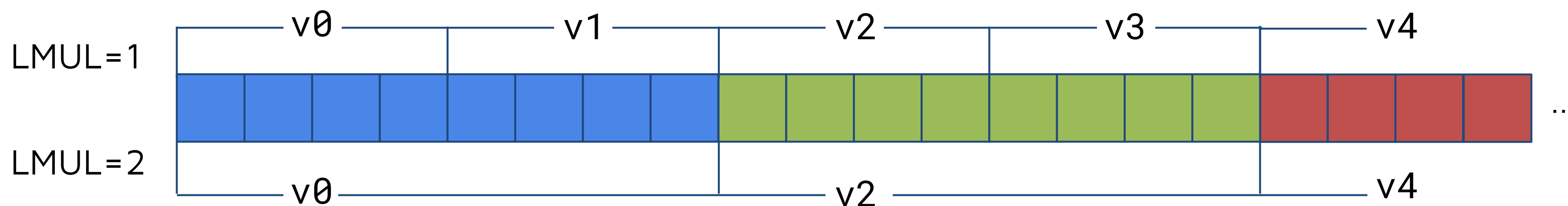
```
vsetivli t0, 5, e32, m1, ta, ma
vadd.vv v1, v2, v3, v0.t
```



Register grouping LMUL

- Controlled by length multiplier **LMUL**, stored in **VTYPE**
- At LMUL=1, can operate on **VLEN / SEW** elements
- At LMUL=2, can operate on **VLEN * 2 / SEW** elements

```
vsetivli t0, 5, e32, m2, ta, ma
vadd.vv v2, v4, v6, v0.t
```



How can we express RVV semantics in LLVM IR?

Plain LLVM IR

```
%v = add <vscale x 2 x i32> %x, %y
```

RVV specific intrinsics

```
%v = call <vscale x 2 x i32> @llvm.riscv.vadd.mask.nxv2i32(
    <vscale x 2 x i32> %passthru, <vscale x 2 x i32> %x,
    <vscale x 2 x i32> %y, <vscale x 2 x i1> %mask,
    i64 %avl, i64 %policy
)
```

Vector predication (VP) intrinsics

```
%v = call <vscale x 2 x i32> @llvm.vp.add.nxv2i32(
   <vscale x 2 x i32> %x, <vscale x 2 x i32> %y, <vscale x 2 x i1> %mask, i32 %evl
)
```

How can we express RVV semantics in LLVM IR?

Plain LLVM IR

```
%v = add <vscale x 2 x i32> %x, %y
```

RVV specific intrinsics

```
%v = call <vscale x 2 x i32> @llvm.riscv.vadd.mask.nxv2i32(
    <vscale x 2 x i32> %passthru, <vscale x 2 x i32> %x,
    <vscale x 2 x i32> %y, <vscale x 2 x i1> %mask,
    i64 %avl, i64 %policy
)
```

Vector predication (VP) intrinsics

```
%v = call <vscale x 2 x i32> @llvm.vp.add.nxv2i32(
   <vscale x 2 x i32> %x, <vscale x 2 x i32> %y, <vscale x 2 x i1> %mask, i32 %evl
)
```

How do we control register grouping?

```
%v = add <vscale x 2 x i32> %x, %y
```

```
vsetvli t0, zero, e32, m1, tu, mu
vadd.vv v0, v1, v2
```

```
%v = add <vscale x 4 x i32> %x, %y
```

```
vsetvli t0, zero, e32, m2, tu, mu
vadd.vv v0, v1, v2
```

```
%v = add <vscale x 8 x i32> %x, %y
```

```
vsetvli t0, zero, e32, m4, tu, mu
vadd.vv v0, v1, v2
```

LLVM IR

```
%v = call <vscale x 2 x i32> @llvm.riscv.vadd.mask.nxv2i32(...)
```

???

MCInst

```
vsetvli t0, a0, e32, m1, ta, ma
vadd.vv v1, v2, v3, v0.t
```

LLVM IR

```
%v = call <vscale x 2 x i32> @llvm.riscv.vadd.mask.nxv2i32(...)
```

SelectionDAG

```
v:nxv2i32 = llvm.riscv.vadd x:nxv2i32, y:nxv2i32, mask:nxv2i1, vl:i32, 0
```

???

MCInst

```
vsetvli t0, a0, e32, m1, ta, ma
vadd.vv v1, v2, v3, v0.t
```

LLVM IR

```
%v = call <vscale x 2 x i32> @llvm.riscv.vadd.mask.nxv2i32(...)
```

SelectionDAG

```
v:nxv2i32 = llvm.riscv.vadd x:nxv2i32, y:nxv2i32, mask:nxv2i1, vl:i32, 0
```

Instruction selection

MachineInstr

```
%v:vr = PseudoVADD_VV_M1_MASK %passthru:vr, %x:vr, %y:vr,
                              %mask:vr, %avl:gpr, 5, 0
```

???

MCInst

```
vsetvli t0, a0, e32, m1, ta, ma
vadd.vv v1, v2, v3, v0.t
```

LLVM IR

```
%v = call <vscale x 2 x i32> @llvm.riscv.vadd.mask.nxv2i32(...)
```

SelectionDAG

```
v:nxv2i32 = llvm.riscv.vadd x:nxv2i32, y:nxv2i32, mask:nxv2i1, vl:i32, 0
```

Instruction selection

MachineInstr

```
%v:vr = PseudoVADD_VV_M1_MASK %passthru:vr, %x:vr, %y:vr,
                               %mask:vr, %avl:gpr, 5, 0
```

RISCVInsertVSETVLI

```
VSETVLI %avl:gpr, e32, m1, 0, implicit-def $vl, implicit-def $vtype
%v:vr = VADD_VV %0:vr(tied-def 0), %x:vr, %y:vr, $v0,
                    implicit $vl, implicit $vtype
```

MCInst

```
vsetvli t0, a0, e32, m1, ta, ma
vadd.vv v1, v2, v3, v0.t
```

RISCVInsertVSETVLI

```
PseudoVLE32_V_M1 ...
PseudoVADD_VI_M1 ...
PseudoVSE32_V_M1 ...
```

→

```
vsetvli zero, a0, e32, m1, tu, mu
vle32.v v1, (a1)
vsetvli zero, a0, e32, m1, tu, mu
vadd.vi v1, v1, 1
vsetvli zero, a0, e32, m1, tu, mu
vse32.v v1, (a1)
```

RISCVInsertVSETVLI

```
PseudoVLE32_V_M1 ...

PseudoVADD_VI_M1 ...

PseudoVSE32_V_M1 ...
```

```
vsetvli zero, a0, e32, m1, tu, mu
vle32.v v1, (a1)

vadd.vi v1, v1, 1

vse32.v v1, (a1)
```

SLP Vectorization

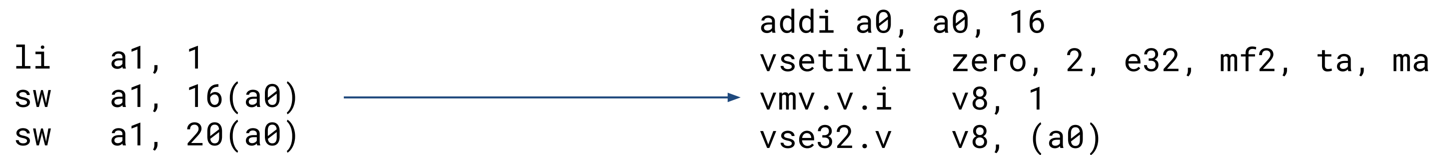
- Vectorizes straight line code to vectorized code
- Generates **fixed-length** vectors:
  - Insert into scalable container type that's guaranteed to fit
  - Legalize to RISCVISD::*_VL node
  - Pass vector length to VL operand
  - Then gets selected as a vector pseudo

```
%p1 = getelementptr i32, ptr %dest, i64 4
store i32 1, ptr %p1
%p2 = getelementptr i32, ptr %dest, i64 5
store i32 1, ptr %p2
```

```
%p1 = getelementptr i32, ptr %dest, i64 4
store <2 x i32> <i32 1, i32 1>, ptr %p
```

SLP Vectorization

- Some things in RVV are more expensive than other targets!



SLP Vectorization

- Some things in RVV are more expensive than other targets!
  - No addressing modes in vector load/stores
  - Overhead of vsetvli not costed for
  - Inserts and extracts can be expensive on larger LMULs
  - Which means constant materialization of vectors can be expensive
- Work done to improve cost model
- Recently enabled by default in LLVM 17 now that it's overall profitable

```
li    a1, 1
sw    a1, 16(a0)
sw    a1, 20(a0)
```

→

```
addi a0, a0, 16
vsetivli  zero, 2, e32, mf2, ta, ma
vmv.v.i   v8, 1
vse32.v   v8, (a0)
```

```
; SVE
movi v0.2s, #1
str  d0, [x0, #16]
```

```c
for (int i = 0; i < n; i++)
  x[i]++;
```

```asm
f:
        blez    a1, .exit
        li      a2, 8
        bgeu    a1, a2, .vector.preheader
        li      a2, 0
        j       .scalar.preheader
.vector.preheader:
        andi    a2, a1, -8
        vsetivli        zero, 8, e32, m2, ta, ma
        mv      a3, a2
        mv      a4, a0
.vector.body:
        vle32.v         v8, (a4)
        vadd.vi         v8, v8, 1
        vse32.v         v8, (a4)
        addi    a3, a3, -8
        addi    a4, a4, 32
        bnez    a3, .vector.body
        beq     a2, a1, .exit
.scalar.preheader:
        slli    a3, a2, 2
        add     a0, a0, a3
        sub     a1, a1, a2
.scalar.body:
        lw      a2, 0(a0)
        addi    a2, a2, 1
        sw      a2, 0(a0)
        addi    a1, a1, -1
        addi    a0, a0, 4
        bnez    a1, .scalar.body
.exit:
        ret
```

Loop Vectorization

- Vectorizing with fixed-length vectors is supported

```c
for (int i = 0; i < n; i++)
  x[i]++;
```

Loop Vectorization

- Vectorizing with fixed-length vectors is supported
- Scalable vectorization is enabled by default

```asm
f:
        blez    a1, .exit
        csrr    a4, vlenb
        srli    a3, a4, 1
        bgeu    a1, a3, .vector.preheader
        li      a2, 0
        j       .scalar.preheader
.vector.preheader:
        srli    a2, a4, 3
        slli    a5, a2, 2
        slli    a2, a2, 32
        sub     a2, a2, a5
        and     a2, a2, a1
        slli    a4, a4, 1
        vsetvli         a5, zero, e32, m2, ta, ma
        mv      a5, a2
        mv      a6, a0
.vector.body:
        vl2re32.v       v8, (a6)
        vadd.vi         v8, v8, 1
        vs2r.v v8, (a6)
        sub     a5, a5, a3
        add     a6, a6, a4
        bnez    a5, .vector.body
        beq     a2, a1, .exit
.scalar.preheader:
        slli    a3, a2, 2
        add     a0, a0, a3
        sub     a1, a1, a2
.scalar.body:
        lw      a2, 0(a0)
        addiw   a2, a2, 1
        sw      a2, 0(a0)
        addi    a1, a1, -1
        addi    a0, a0, 4
        bnez    a1, .scalar.body
.exit:
        ret
```

```
for (int i = 0; i < n; i++)
  x[i]++;
```

```
f:
        blez    a1, .exit
        csrr    a4, vlenb
        srli    a3, a4, 1
        bgeu    a1, a3, .vector.preheader
        li      a2, 0
        j       .scalar.preheader
.vector.preheader:
        srli    a2, a4, 3
        slli    a5, a2, 2
        slli    a2, a2, 32
        sub     a2, a2, a5
        and     a2, a2, a1
        slli    a4, a4, 1
        vsetvli         a5, zero, e32, m2, ta, ma
        mv      a5, a2
        mv      a6, a0
.vector.body:
        vl2re32.v       v8, (a6)
        vadd.vi         v8, v8, 1
        vs2r.v  v8, (a6)
        sub     a5, a5, a3
        add     a6, a6, a4
        bnez    a5, .vector.body
        beq     a2, a1, .exit
.scalar.preheader:
        slli    a3, a2, 2
        add     a0, a0, a3
        sub     a1, a1, a2
.scalar.body:
        lw      a2, 0(a0)
        addiw   a2, a2, 1
        sw      a2, 0(a0)
        addi    a1, a1, -1
        addi    a0, a0, 4
        bnez    a1, .scalar.body
.exit:
        ret
```

Loop Vectorization

- Vectorizing with fixed-length vectors is supported
- Scalable vectorization is enabled by default
- Uses LMUL=2 by default
  - Could be smarter and increase it, but need to account for register pressure

```
for (int i = 0; i < n; i++)
  x[i]++;
```

Loop Vectorization

- Vectorizing with fixed-length vectors is supported
- Scalable vectorization is enabled by default
- Uses LMUL=2 by default
    - Could be smarter and increase it, but need to account for register pressure
- By default scalar epilogue is emitted…

```
f:
        blez    a1, .exit
        csrr    a4, vlenb
        srli    a3, a4, 1
        bgeu    a1, a3, .vector.preheader
        li      a2, 0
        j       .scalar.preheader
.vector.preheader:
        srli    a2, a4, 3
        slli    a5, a2, 2
        slli    a2, a2, 32
        sub     a2, a2, a5
        and     a2, a2, a1
        slli    a4, a4, 1
        vsetvli         a5, zero, e32, m2, ta, ma
        mv      a5, a2
        mv      a6, a0
.vector.body:
        vl2re32.v       v8, (a6)
        vadd.vi         v8, v8, 1
        vs2r.v v8, (a6)
        sub     a5, a5, a3
        add     a6, a6, a4
        bnez    a5, .vector.body
        beq     a2, a1, .exit
.scalar.preheader:
        slli    a3, a2, 2
        add     a0, a0, a3
        sub     a1, a1, a2
.scalar.body:
        lw      a2, 0(a0)
        addiw   a2, a2, 1
        sw      a2, 0(a0)
        addi    a1, a1, -1
        addi    a0, a0, 4
        bnez    a1, .scalar.body
.exit:
        ret
```

```
for (int i = 0; i < n; i++)
  x[i]++;
```

```
f:
        blez    a1, .exit
        li      a2, 0
        csrr    a5, vlenb
        srli    a3, a5, 1
        neg     a4, a3
        add     a6, a3, a1
        addi    a6, a6, -1
        and     a4, a6, a4
        slli    a5, a5, 1
        vsetvli         a6, zero, e64, m4, ta, ma
        vid.v   v8
.vector.body:
        vsetvli         zero, zero, e64, m4, ta, ma
        vsaddu.vx       v12, v8, a2
        vmsltu.vx       v0, v12, a1
        vle32.v         v12, (a0), v0.t
        vsetvli         zero, zero, e32, m2, ta, ma
        vadd.vi         v12, v12, 1
        vse32.v         v12, (a0), v0.t
        add     a2, a2, a3
        add     a0, a0, a5
        bne     a4, a2, .vector.body
.exit:
        ret
```

Loop Vectorization

- Vectorizing with fixed-length vectors is supported
- Scalable vectorization is enabled by default
- Uses LMUL=2 by default
  - Could be smarter and increase it, but need to account for register pressure
- By default scalar epilogue is emitted…
  - But predicated tail folding can be enabled with -prefer-predicate-over-epilogue

```c
for (int i = 0; i < n; i++)
  x[i]++;
```

```asm
f:
        blez    a1, .exit
.vector.body:
        vsetvli         t0, a1, e32, m2, ta, ma
        vle32.v         v12, (a0)
        vadd.vi         v12, v12, 1
        vse32.v         v12, (a0)
        sub     a1, a1, t0
        add     a0, a0, t0
        bnez    a1, .vector.body
.exit:
        ret
```

Loop Vectorization

- Vectorizing with fixed-length vectors is supported
- Scalable vectorization is enabled by default
- Uses LMUL=2 by default
  - Could be smarter and increase it, but need to account for register pressure
- By default scalar epilogue is emitted…
  - But predicated tail folding can be enabled with -prefer-predicate-over-epilogue
  - Work is underway to perform **tail folding via VL** (D99750)
    - Via VP intrinsics

What next?

- Teach loop vectorizer to dynamically select an LMUL
- Round out vector predication support
  - Lift InstCombine/DAGCombine to handle VP intrinsics
- Handle scalable interleaved accesses with more than 2 groups
  - Take advantage of segmented loads/stores: vlseg/vsseg
- Data dependent loop exits
  - Take advantage of fault-only-first loads: vle32ff.v
- Use VL to allow SLP to vectorize non-power-of-2 VFs

igalia