

2024 RISC-V Summit China

RVV Auto-Vectorization in GCC

Pan Li – Intel

Liu, Hongtao - Intel



Legal Notices and Disclaimers

Statements in this document that refer to future plans or expectations are forward-looking statements. These statements are based on current expectations and involve many risks and uncertainties that could cause actual results to differ materially from those expressed or implied in such statements. For more information on the factors that could cause actual results to differ materially, see our most recent earnings release and SEC filings at www.intc.com.

All product plans and roadmaps are subject to change without notice. Any forecasts of goods and services needed for Intel's operations are provided for discussion purposes only. Intel will have no liability to make any purchase in connection with forecasts published in this document. Code names are often used by Intel to identify products, technologies, or services that are in development and usage may change over time. No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

© Intel Corporation. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others. This document contains information on products and/or processes in development.

Agenda

- [Scalable/Fixed](#)
- [Advanced VSETVL](#)
- [Dynamic LMUL](#)
- [Conditional](#)
- [Rounding](#)
- [Advanced FRM](#)
- [Early Exit \(GCC-15\)](#)
- [Saturation ALU \(GCC-15\)](#)
- [More in GCC-15 for Auto-Vectorization](#)

Scalable (default) – by [RiVAI](#), [Ventana Micro](#) and [Intel](#)

-mrvv-vector-bits=**scalable**

Any vlen is a multiple of zvl bits.

```
1. void
2. vec_shift (int64_t *out, int64_t *a,
3.           uint32_t size)
4. {
5.     for (uint32_t i = 0; i < size; i++)
6.         out[i] = a[i] << 7;
7. }
```

```
10  vec_shift:
11  .LFB0:
12      .cfi_startproc
13      beq a2,zero,.L12
14      addiw a5,a2,-1
15      li a4,4
16      bleu a5,a4,.L3
17      addi a4,a1,8
18      sub a4,a0,a4
19      csrr a5,vlenb
20      addi a5,a5,-16
21      bleu a4,a5,.L3
22      slli a2,a2,32
23      srli a2,a2,32
24  .L4:
25      vsetvli a5,a2,e64,m1,ta,ma
26      slli a4,a5,3
27      vle64.v v1,0(a1)
28      vsll.vi v1,v1,7
29      vse64.v v1,0(a0)
30      add a1,a1,a4
31      add a0,a0,a4
32      sub a2,a2,a5
33      bne a2,zero,.L4
34      ret
```

Fixed - by [RiVAI](#), [Ventana Micro](#) and [Intel](#)

-mrvv-vector-bits=**zvl**

The vlen is exactly same as zvl bits.

```
1. void
2. vec_shift (int64_t *out, int64_t *a,
3.            uint32_t size)
4. {
5.     for (uint32_t i = 0; i < size; i++)
6.         out[i] = a[i] << 7;
7. }
```

```
10  vec_shift:
11  .LFB0:
12      .cfi_startproc
13      beq a2,zero,.L12
14      addiw a5,a2,-1
15      li a4,4
16      bleu a5,a4,.L3
17      addi a5,a1,8
18      beq a0,a5,.L3
19      slli a2,a2,32
20      srli a2,a2,32
21  .L4:
22      vsetvli a5,a2,e64,m1,ta,ma
23      slli a4,a5,3
24      vle64.v v1,0(a1)
25      vsll.vi v1,v1,7
26      vse64.v v1,0(a0)
27      add a1,a1,a4
28      add a0,a0,a4
29      sub a2,a2,a5
30      bne a2,zero,.L4
31      ret
```

Advanced VSETVL (default on) - by [RiVAI](#)

```
1. void foo (int8_t * restrict in, int8_t * restrict out,  
2.           int n, int cond)  
3. {  
4.     size_t vl = 101; double f = -0.94;  
5.     vfloat64m4_t v2, v3, v4;  
6.     if (cond > 231) {  
7.         v2 = __riscv_vle64_v_f64m4 (in, vl);  
8.         f = __riscv_vfmv_f_s_f64m4_f64 (v2);  
9.     }  
10.    for (size_t i = 0; i < n; i++) {  
11.        v3 = __riscv_vle64_v_f64m4 (in + i + 32, vl);  
12.        v4 = __riscv_vle64_v_f64m4 (in + i + 64, vl);  
13.        v4 = __riscv_vfmacc_vf_f64m4 (v4, f, v3, vl);  
14.        __riscv_vse64_v_f64m4 (out + 128, v4, vl);  
15.    }  
16. }
```

```
10  foo:  
11  .LFB0:  
12      .cfi_startproc  
13      li    a5,231  
14      bleu  a3,a5,.L5  
15      li    a5,101  
16      vsetvli zero,a5,e64,m4,ta,ma  
17      vle64.v v4,0(a0)  
18      vfmv.f.s fa5,v4  
19  .L2:  
20      beq a2,zero,.L10  
21      addi   a5,a0,100  
22      addi   a1,a1,228  
23      addi   a2,a2,100  
24      add a0,a0,a2  
25      li    a4,101  
26      vsetvli zero,a4,e64,m4,ta,ma  
27  .L4:  
28      vle64.v v8,0(a5)  
29      addi   a3,a5,256  
30      vle64.v v4,0(a3)  
31      vfmacc.vf v4,fa5,v8  
32      vse64.v v4,0(a1)  
33      addi   a5,a5,1  
34      addi   a1,a1,1  
35      bne a5,a0,.L4  
36  .L10:  
37      ret
```

Dynamic LMUL - by [RiVAI](#)

-mrvv-max-lmul=**m1** (default)

-mrvv-max-lmul=**dynamic**

1. void

2. foo (int32_t *__restrict a,

3. int32_t *__restrict b, int n)

4. {

5. for (int i = 0; i < n; i++)

6. a[i] = a[i] + b[i];

7. }

```
10  foo:
11  .LFB0:
12      .cfi_startproc
13      ble a2,zero,.L5
14      mv  a4,a0
15  .L3:
16      vsetvli a5,a2,e32,m1,ta,ma
17      slli  a3,a5,2
18      vle32.v v2,0(a0)
19      vle32.v v1,0(a1)
20      vadd.vv v1,v1,v2
21      vse32.v v1,0(a4)
```

```
10  foo:
11  .LFB0:
12      .cfi_startproc
13      ble a2,zero,.L5
14      mv  a4,a0
15  .L3:
16      vsetvli a5,a2,e32,m8,ta,ma
17      slli  a3,a5,2
18      vle32.v v16,0(a0)
19      vle32.v v8,0(a1)
20      vadd.vv v8,v8,v16
21      vse32.v v8,0(a4)
```

Conditional (default on) - by [RiVAI](#) and [Ventana Micro](#)

```
1. void
2. vec_fmax (double *__restrict a,
3.           double *__restrict b,
4.           int64_t *pred, int n)
5. {
6.     for (int i = 0; i < n; i++)
7.     {
8.         if (pred[i] != 1)
9.             a[i] = __builtin_fmaxf64 (b[i], 3.0);
10.        else
11.            a[i] = b[i];
12.    }
13. }
```

```
10  vec_fmax:
11  .LFB0:
12      .cfi_startproc
13      ble a3,zero,.L6
14      lui a5,%hi(.LC0)
15      fld fa5,%lo(.LC0)(a5)
16      vsetvli a5,zero,e64,m1,ta,ma
17      vfmv.v.f v2,fa5
18  .L3:
19      vsetvli a5,a3,e64,m1,ta,mu
20      slli a4,a5,3
21      vle64.v v0,0(a2)
22      vle64.v v1,0(a1)
23      vmsne.vi v0,v0,1
24      vfmv.vv v1,v1,v2,v0.t
25      vse64.v v1,0(a0)
26      add a2,a2,a4
27      add a1,a1,a4
28      add a0,a0,a4
29      sub a3,a3,a5
30      bne a3,zero,.L3
31  .L6:
32      ret
```


Rounding - by [Intel](#)

-ffast-math

round/ceil/floor/rint/trunc/nearbyint

```
1. void
2. vec_lround (long *out, float *a, int n)
3. {
4.     for (int i = 0; i < n; i++)
5.         out[i] = __builtin_lroundf (a[i]);
6. }
```

```
10  vec_lround:
11  .LFB0:
12      .cfi_startproc
13      frrm    a3
14      ble a2,zero,.L5
15      fsrmi   4
16  .L3:
17      vsetvli a5,a2,e32,mf2,ta,ma
18      vle32.v v2,0(a1)
19      vfwcvt.x.f.v v1,v2
20      vse64.v v1,0(a0)
21      slli    a4,a5,2
22      add a1,a1,a4
23      slli    a4,a5,3
24      add a0,a0,a4
25      sub a2,a2,a5
26      bne a2,zero,.L3
27  .L5:
28      fsrm    a3
29      ret
```

Advanced FRM (default on) - by [Intel](#)

```
1. void
2. foo (float *in, float *out, int n, int cond, size_t vl)
3. {
4.     vfloat32m1_t v, result;
5.     if (cond < 0xde)
6.     {
7.         v = __riscv_vle32_v_f32m1 (in + 16, vl);
8.         result = __riscv_vfadd_vv_f32m1_rm (v, v, 3, vl);
9.         __riscv_vse32_v_f32m1 (out + 16, result, vl);
10.    }
11.    for (int i = 0; i < n; i++)
12.    {
13.        v = __riscv_vle32_v_f32m1 (in + i + 200, vl);
14.        result = __riscv_vfadd_vv_f32m1_rm (v, v, 3, vl);
15.        __riscv_vse32_v_f32m1 (out + i + 200, result, vl);
16.    }
17.}
```

```
10  foo:
11  .LFB0:
12      .cfi_startproc
13      frrm    a6          backup
14      li     a5,221
15      bgt    a3,a5,.L9
16      vsetvli zero,a4,e32,m1,ta,ma
17      addi   a5,a0,64
18      vle32.v v1,0(a5)
19      fsrmi   3
20      vfadd.vv v1,v1,v1
21      addi   a5,a1,64
22      vse32.v v1,0(a5)
23  .L2:
24      ble    a2,zero,.L6
25      addi   a5,a0,800
26      addi   a1,a1,800
27      slli   a2,a2,2
28      addi   a2,a2,800
29      add    a0,a0,a2
30      fsrmi   3
31  .L4:
32      vle32.v v1,0(a5)
33      vfadd.vv v1,v1,v1
34      vse32.v v1,0(a1)
35      addi   a5,a5,4
36      addi   a1,a1,4
37      bne    a5,a0,.L4
38  .L6:
39      fsrm    a6          restore
40      ret
41  .L9:
42      vsetvli zero,a4,e32,m1,ta,ma
43      j      .L2
```

Early Exit (GCC-15) – by [RiVAI](#) and [Intel](#)

```
1. #define N 901
2. unsigned vect_a[N];
3. unsigned vect_b[N];
4. void test (unsigned x, int n)
5. {
6.     for (int i = 0; i < n; i++)
7.     {
8.         vect_b[i] = x + i;
9.         if (vect_a[i] > x)
10.            break;
11.         vect_a[i] = x;
12.     }
13. }
```

```
29      .L4:
30          vse32.v v5,0(a4)
31          add t3,t3,a7
32          add a4,a4,a7
33          vse32.v v4,0(a6)
34          vsetvli a2,zero,e32,m1,ta,ma
35          vadd.vv v3,v2,v3
36          add a6,a6,a7
37          beq a3,zero,.L14
38      .L5:
39          vsetvli a5,a3,e32,m1,ta,ma
40          vle32.v v1,0(t3)
41          vadd.vv v5,v3,v4
42          vsetvli a2,zero,e32,m1,ta,ma
43          vmv.v.x v2,a5
44          vsetvli zero,a5,e32,m1,ta,ma
45          slli a7,a5,2
46          vmsltu.vv v1,v4,v1
47          sub a3,a3,a5
48          vcpop.m t5,v1
49          beq t5,zero,.L4
50          vmv.x.s a4,v3
```

Saturation ALU (GCC-15) - by [Intel](#)

Sub

```
1. void
2. zip (uint16_t *x, uint32_t b, unsigned n)
3. {
4.     uint32_t a;
5.     uint16_t *p = x;
6.     do {
7.         a = *--p;
8.         *p = (uint16_t)(a >= b ? a - b : 0);
9.     } while (--n);
10. }
```

```
24      vid.v    v2
25      li      a4,-1
26      vrsb.vx  v2,v2,a6
27      vnclipu.wi v4,v4,0
28      .L3:
29      vle16.v  v3,0(a3)
30      mv      a6,a4
31      addw    a4,a4,t1
32      vrgather.vv v1,v3,v2
33      vssubu.vv  v1,v1,v4
34      vrgather.vv v3,v1,v2
35      vse16.v  v3,0(a3)
36      sub     a3,a3,a7
37      bgtu    t3,a4,.L3
```

Saturation ALU (GCC-15) - by [Intel](#)

Add

```
1. void
2. vec_sat_add (uint64_t *out, uint64_t *a,
3.             uint64_t *b, uint32_t size)
4. {
5.     for (uint32_t i = 0; i < size; i++)
6.     {
7.         if ((a[i] + b[i]) >= a[i])
8.             out[i] = a[i] + b[i];
9.         else
10.            out[i] = -1;
11.     }
12. }
```

```
26      .L5:
27          vsetvli a5,a3,e64,m1,ta,ma
28          vle64.v v1,0(a1)
29          vle64.v v2,0(a2)
30          slli    a4,a5,3
31          sub     a3,a3,a5
32          add     a1,a1,a4
33          add     a2,a2,a4
34          vsaddu.vv v1,v1,v2
35          vse64.v v1,0(a0)
36          add     a0,a0,a4
37          bne     a3,zero,.L5
38          ret
```

Saturation ALU (GCC-15) - by [Intel](#)

Truncate

```
1. void
2. vec_sat_trunc (uint32_t *out, uint64_t *a,
3.               uint32_t size)
4. {
5.     for (uint32_t i = 0; i < size; i++)
6.     {
7.         uint64_t x = a[i];
8.         bool overflow = x > (uint64_t)(uint32_t)(-1);
9.         if (overflow)
10.            out[i] = -1;
11.         else
12.            out[i] = (uint32_t)x;
13.     }
14. }
```

```
15 .L3:
16     vsetvli a5,a2,e32,mf2,ta,ma
17     vle64.v v1,0(a1)
18     slli    a3,a5,3
19     slli    a4,a5,2
20     sub     a2,a2,a5
21     add     a1,a1,a3
22     vnclipu.wi v1,v1,0
23     vse32.v v1,0(a0)
24     add     a0,a0,a4
25     bne     a2,zero,.L3
```

More in GCC-15

- Stride Load/Store.
- Vector Register Overlap.
- VLS Calling Convention.



RISE

RISC-V Software Ecosystem

- <https://riseproject.dev>

RISE is focused on positive and transparent collaborations with upstream projects to deliver commercial-ready software for various use cases

How: Align on highest priorities & avoid (accidental) duplication of work

Goal: Accelerate open source SW for RISC-V architecture

<https://www.intel.com/content/www/us/en/developer/articles/community/rising-to-the-challenge-risc-v-software-readiness.html>

Finding more interesting topics from Intel on RISC-V summit China 2024

Topic	When & Where
UXL 软件栈和 RISC-V 的初步探索	August 22 16:45 主会场A
LLVM 工具链 RISC-V 构建实现及其性能优化现状分析与未来展望	August 23 9:40 主会场A
GCC RVV 自动向量化及其应用	August 23 10:00 主会场A
Enhancing RISC-V Security with SBI Secure Service APIs	August 23 10:40 主会场B
Enabling Hardware Sampling Based PGO for RISC-V Platform	August 23 11:40 主会场A
利用 WASM 技术解决多种 ISA 的挑战	August 23 14:20 主会场B
HVP: Hardware Accelerated RISC-V Android Emulator	August 23 14:50 主会场A
Leverage BRS standard to improve RISC-V SW compatibility	August 23 17:30 主会场A
Soft-ISA: kernel built-in emulation engine to extend RISC-V silicon ISA capability	August 23 17:40 主会场A

The Intel logo is centered on a solid blue background. It features the word "intel" in a white, lowercase, sans-serif font. A small, light blue square is positioned above the first vertical stroke of the letter 'i'. To the right of the word "intel" is a small white registered trademark symbol (®).

intel®