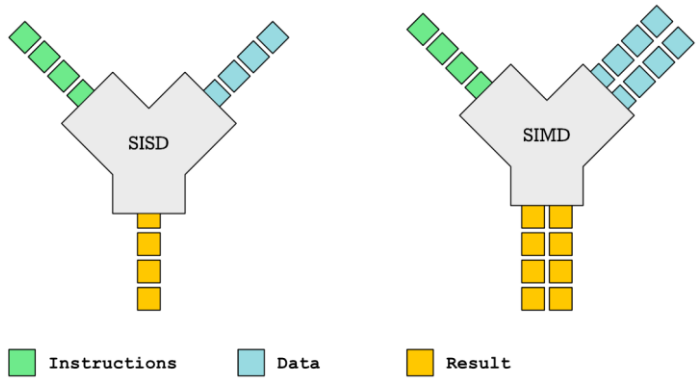
ARM SVE指令优化介绍

华为 陈正

HUAWEI

Scalable Vector Extension

- SVE和Neon都属于 SIMD (Single Instruction Multiple Data)指令集




```
for(i=0;i<n;++i)
  a[i] = a[i] * 2.0;
```

```
for(i=0;i<n;i+=4)
  a[i:i+3] = a[i:i+3] * v_2.0;
```

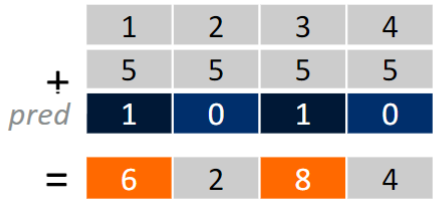
- 关键特性
  - > Scalable vectors z0 ~ z31, 128 ~ 2048 bit
  - > Per-lane predication
  - > Gather-load and scatter-store
  - > Speculative vectorization
  - > Horizontal Operations

- 优势
  - > 同一个二进制，适配所有长度的SVE平台
  - > 提供更多向量化的机会
  - > 简化控制流，避免if/else分支
  - > 提供更宽的寄存器位宽

HUAWEI

关键特性1：Per-lane predication

- Per-lane predication ：对每个通道的指令进行掩码操作，确定通道有效性




```
for (i = 0; i < N; ++i)
    a[i] = b[i] + c[i];
```

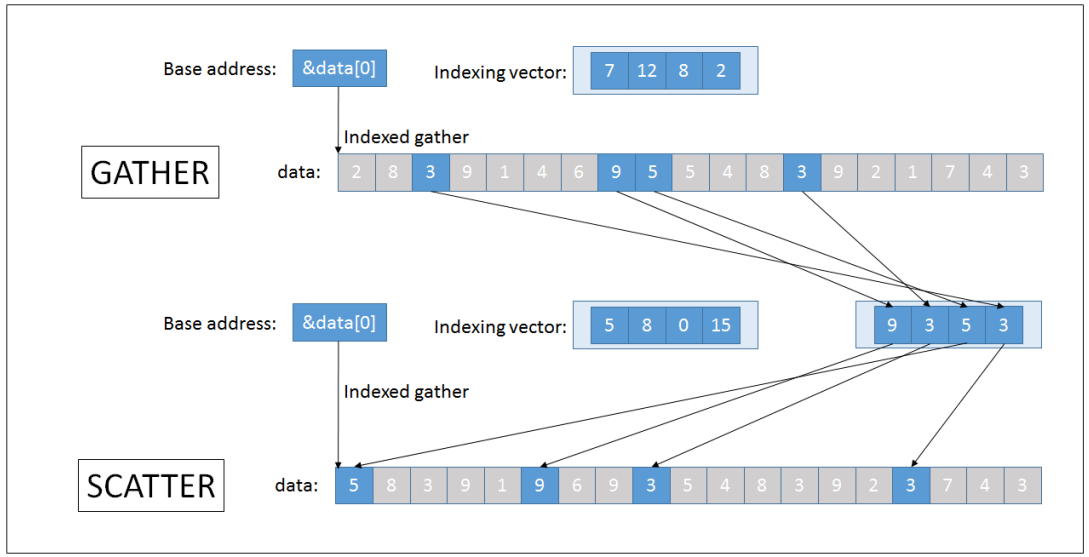
```
// vector loop:
for (i = 0; i < N - 3; i += 4) {
      int32x4_t vb = vld1q_s32(b + i);
      int32x4_t vc = vld1q_s32(c + i);
      int32x4_t va = vaddq_s32(vb, vc);
      vst1q_s32(a + i, va);
}
// loop tail
for (; i < N; ++i)
       a[i] = b[i] + c[i];
}
```

```
for(int i= 0 ; i< N; i+= svcntw())
{
    svbool_t Pg= svwhilelt_b32(i, N);
    svfloat32_t vb= svld1(Pg, &b[i]);
    svfloat32_t vc= svld1(Pg, &c[i]);
    va= svadd_z(Pg, vb, vc);
    svst1(Pg, &a[i], va);
}
```

HUAWEI

关键特性2：Gather-load and scatter-store

- 非连续地址访存，把访问的地址的offset存放在vector中
  - LD1B { <Zt>.D }, <Pg>/Z, [<Xn|SP>, <Zm>.D, <mod>]
  - ST1B { <Zt>.D }, <Pg>, [<Xn|SP>, <Zm>.D, <mod>]



关键特性2：Gather-load

```
void gather_copy(const float * restrict data,
                 const int * restrict index,
                  float * restrict result, int N) {
    for (int i = 0; i < N; ++i) {
        result[i] = data[index[i]];
    }
}
```

```
.LBB0_4:
        ldpsw   x13, x14, [x10, #-4]
        subs    x12, x12, #2
        add     x10, x10, #8
        ldr     s0, [x0, x13, lsl #2]
        ldr     s1, [x0, x14, lsl #2]
        stp     s0, s1, [x11, #-4]
        add     x11, x11, #8
        b.ne    .LBB0_4
```

```
.LBB0_4:
        ldr     z0, [x10]
        subs    x12, x12, #8
        add     x10, x10, #32
        ld1w    { z0.s }, p0/z, [x0,
                 z0.s, sxtw #2]
        str     z0, [x11]
        add     x11, x11, #32
        b.ne    .LBB0_4
```

https://godbolt.org/z/sPrc7djjf

关键特性2：Scatter-store

```
void scatter_store(float * restrict output,
                   const int * restrict index,
                    const float * restrict values, int N) {
    for (int i = 0; i < N; ++i) {
        output[index[i]] = values[i];
    }
}
```

```
.LBB0_4:
        ldpsw   x13, x14, [x11, #-4]
        ldp     s0, s1, [x10, #-4]
        subs    x12, x12, #2
        add     x10, x10, #8
        add     x11, x11, #8
        str     s0, [x0, x13, lsl #2]
        str     s1, [x0, x14, lsl #2]
        b.ne    .LBB0_4
```

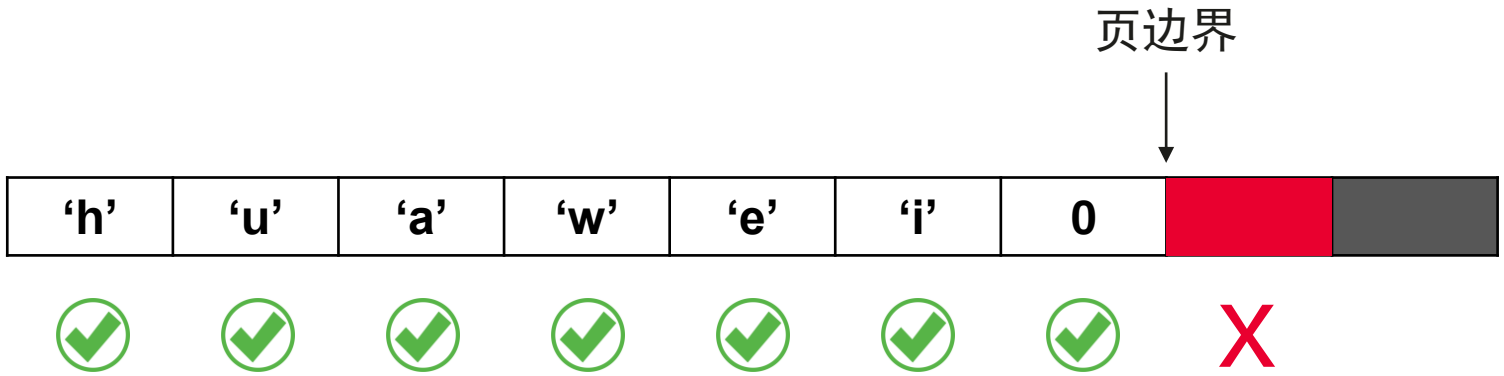
```
.LBB0_4:
        ldr     z0, [x10]
        ldr     z1, [x11]
        subs    x12, x12, #8
        add     x11, x11, #32
        add     x10, x10, #32
        st1w    { z0.s }, p0, [x0, z1.s, sxtw
                 #2]
        b.ne    .LBB0_4
```

https://godbolt.org/z/axd331q5e

关键特性3：Speculative vectorization

```
int strlen(const char *s) {
  const char *e = s;
  while (*e) e++;
  return e - s;
}
```

```
strlen:
        mov     x8, x0
        mov     w0, #-1
.LBB0_1:
        ldrb    w9, [x8], #1
        add     w0, w0, #1
        cbnz    w9, .LBB0_1
        ret
```



SIMD宽load可能会导致内存访问crash！

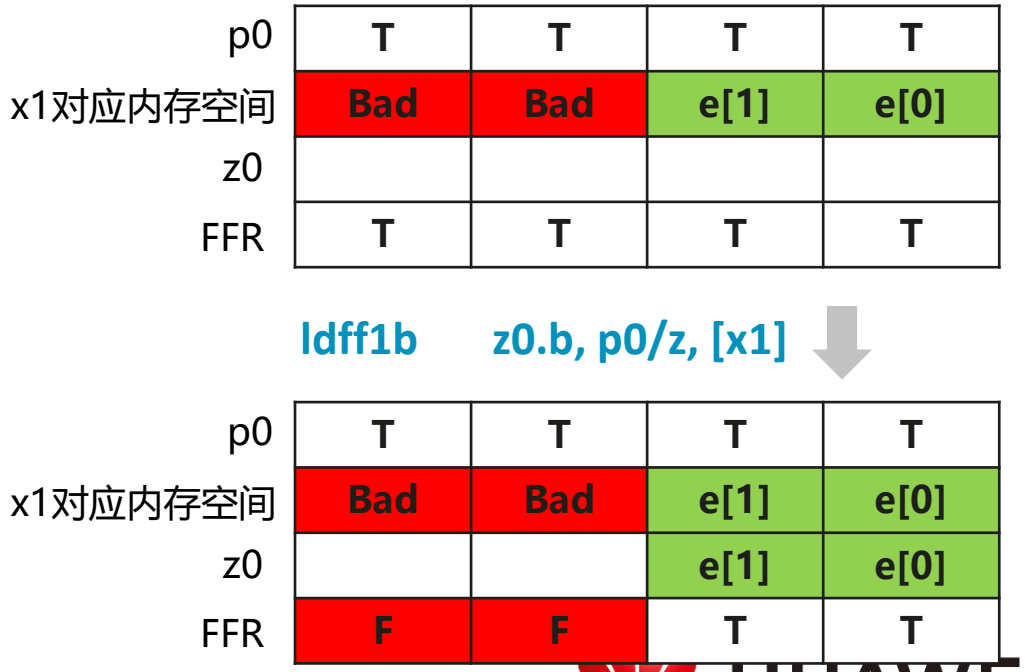
HUAWEI

关键特性3：Speculative vectorization

```
int strlen(const char *s) {
  const char *e = s;
  while (*e) e++;
  return e - s;
}
```

- First Fault Register (FFR) is a special predicate register, which is set by the first-fault load and store instructions, to indicate how successful the load and store operation for each element is. FFR is designed to support speculative memory accesses which make the vectorization, in many situations, easier and safer.

```
strlen:
        mov     x1, x0                  // e=s
        ptrue   p0.b                    // p0=true
.loop:
        setffr                          // ffr=true
        ldff1b  z0.b, p0/z, [x1]        // p0:z0=ldff(e)
        rdffr   p1.b, p0/z              // p0:p1=ffr
        cmpeq   p2.b, p1/z, z0.b, #0    // p1:p2=(*e==0)
        brkbs   p2.b, p1/z, p2.b        // p1:p2=until(*e==0)
        incp    x1, p2.b                // e+=popcnt(p2)
        b.last  .loop                   // last active=>!break
        sub     x0, x1, x0              // return e-s
        ret
```

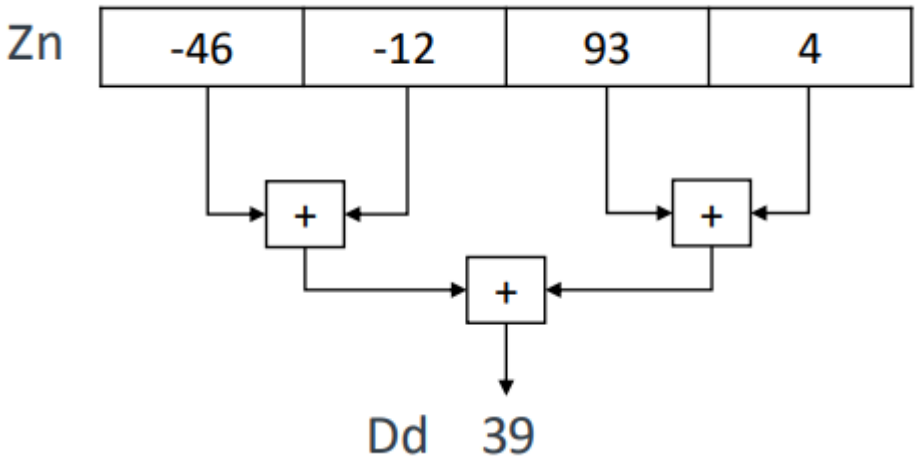


HUAWEI

关键特性4：Horizontal Operations

- 操作跨向量寄存器的不同的lane
- 加、最大值、最小值，逻辑与、逻辑或、逻辑异或等

```
SADDV <Dd>, <Pg>, <Zn.T>
```



HUAWEI

关键特性4：Horizontal Operations

```
double ddot (double *a, double *b, int n)
{
    double sum = 0.0;
    for ( int i = 0; i < n; i++ ) {
        sum += a[i] * b[i];
    }
    return sum;
}
```

```
.LBB0_5:
        ldp     q1, q4, [x10, #-16]
        subs    x12, x12, #4
        ldp     q2, q3, [x11, #-16]
        add     x10, x10, #32
        add     x11, x11, #32
        fmul    v1.2d, v2.2d, v1.2d
        mov     d2, v1.d[1]
        fadd    d0, d0, d1
        fmul    v1.2d, v3.2d, v4.2d
        fadd    d0, d0, d2
        mov     d2, v1.d[1]
        fadd    d0, d0, d1
        fadd    d0, d0, d2
        b.ne    .LBB0_5
```

```
#include <arm_sve.h>
double ddot (double *a, double *b, int n) {
    svfloat64_t svsum = svdup_f64(0.0);
    svbool_t pg;
    svfloat64_t sva, svb;
    for (int i = 0; i < n; i += svcntd()) {
        pg = svwhilelt_b64(i, n);
        sva = svld1_f64(pg, &a[i]);
        svb = svld1_f64(pg, &b[i]);
        svsum = svmla_f64_m(pg, svsum, sva, svb);
    }
    return svaddv_f64(svptrue_b64(), svsum);
}
```

```
.LBB0_2:
        whilelt p0.d, w8, w2
        sxtw    x10, w8
        add     w8, w8, w9
        cmp     w8, w2
        ld1d    { z1.d }, p0/z, [x0, x10, lsl #3]
        ld1d    { z2.d }, p0/z, [x1, x10, lsl #3]
        fmla    z0.d, p0/m, z1.d, z2.d
        b.lt    .LBB0_2
        ptrue   p0.d
        faddv   d0, p0, z0.d
```

HUAWEI

Thank you.

把数字世界带入每个人、每个家庭、
每个组织，构建万物互联的智能世界。

Bring digital to every person, home and organization for a fully connected, intelligent world.

HUAWEI