SC18 Arm SVE Hackathon

Introducing the Armv8-A Scalable Vector Extension

Nov 15th, 2018

Why is it called Scalable Vector Extension?

- SVE does not mandate a single, fixed vector length
  - Vector Length (VL) is hardware implementation choice of 128 to 2048 bits.
  - Vector Length Agnostic (VLA) programming paradigm adapts to the available vector length.

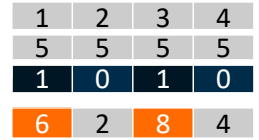
- SVE begins to address traditional barriers to auto-vectorization
  - Compilers often cannot vectorize due to intra-vector control and data dependencies.
  - Software-managed speculative vectorization allows more loops to be vectorized by a compiler.

Introducing the Scalable Vector Extension (SVE)

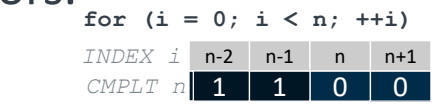
A vector extension to the Armv8-A architecture with some major new features

- **Gather-load and scatter-store**
  - Loads a single register from several non-contiguous memory locations.
  - Enables vectorization of complex data structures with non-linear access patterns.

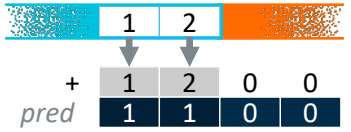
- **Per-lane predication**
  - Operations work on individual lanes under control of a predicate register.
  - Enables vectorization of complex, nested control code containing side effects.



- **Predicate-driven loop control and management**
  - Eliminate loop heads and tails and other overhead by processing partial vectors.
  - Reduces vectorization overhead relative to scalar code.



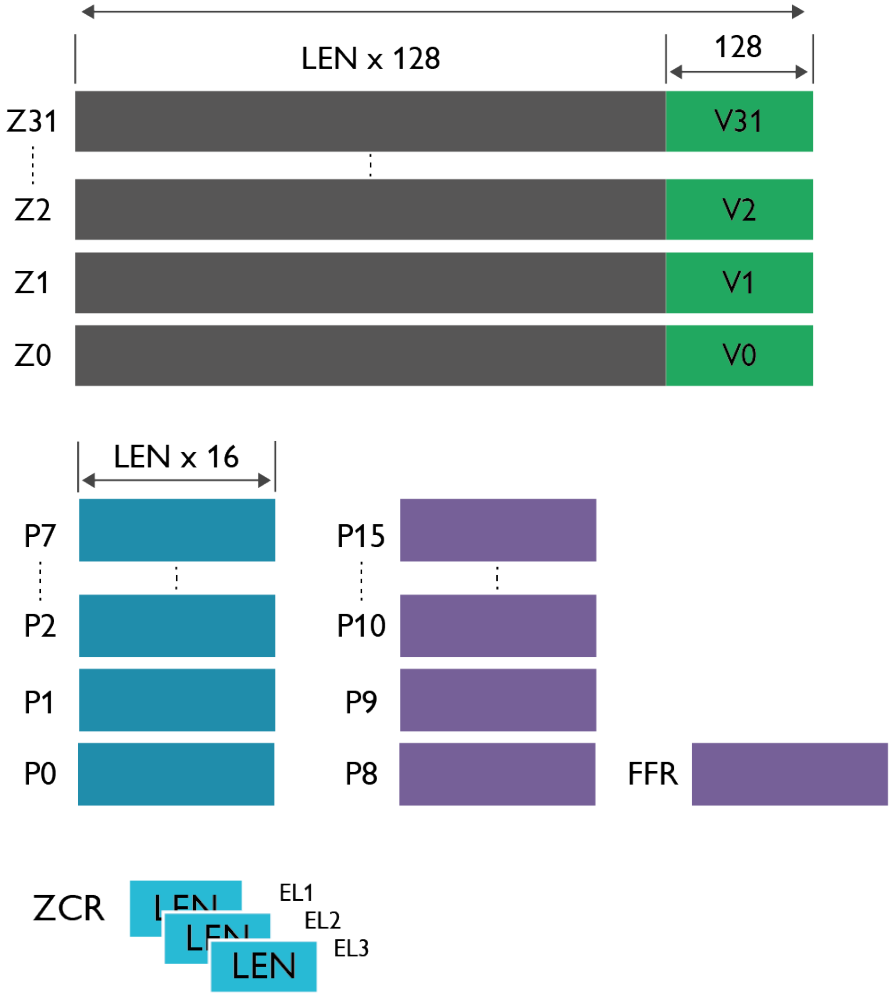
- **Vector partitioning and software-managed speculation**
  - First-faulting vector load instructions allow memory accesses to cross into invalid pages.



arm

New SVE architectural state

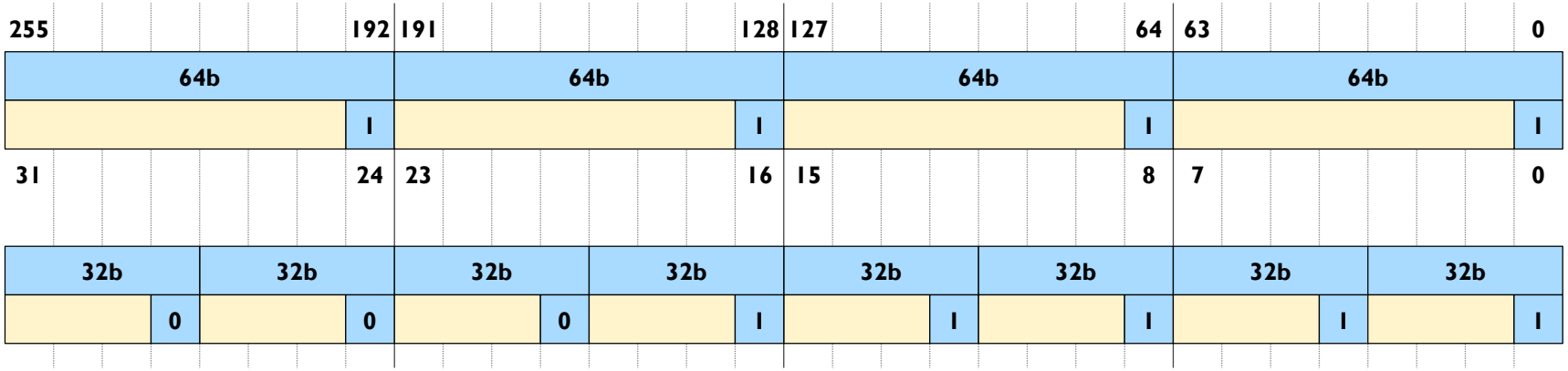
- Scalable vector registers
  - Z0-Z31 extending NEON's V0-V31.
  - Packed DP, SP & HP floating-point elements.
  - Packed 64, 32, 16 & 8-bit integer elements.
- Scalable predicate registers
  - P0-P7 governing predicates for load/store/arithmetic.
  - P8-P15 additional predicates for loop management.
  - FFR first fault register for speculation.
- Scalable vector control registers
  - ZCR_ELx vector length (LEN=1..16).
  - For exception/privilege levels EL1 to EL3.



arm

Predication

- 16 predicate registers defined (P0-P15)
  - 1 bit per lane ∴ 1 predicate bit per 8 vector bits (lowest predicate bit per lane is significant)
- Active elements update destination
  - inactive lanes leave destination unchanged or set to 0's
  - power-saving opportunities
- ≈6% SVE instructions act on and return predicates → "Predicate ALU"



arm

Scalar strlen()

```
// ----------------------------------------
//      int strlen(const char *s) {
//          const char *e = s;
//          while (*e) e++;
//          return e - s;
//      }
// ----------------------------------------
// x0 = s

// Unoptimized A64 scalar strlen
strlen:
     mov       x1, x0              // e=s
.loop:
     ldrb      x2, [x1],#1         // x2=*e++
     cbnz      x2, .loop           // while(*e)
.done:
     sub       x0, x1, x0          // e-s
     sub       x0, x0, #1          // return e-s-1
     ret
```

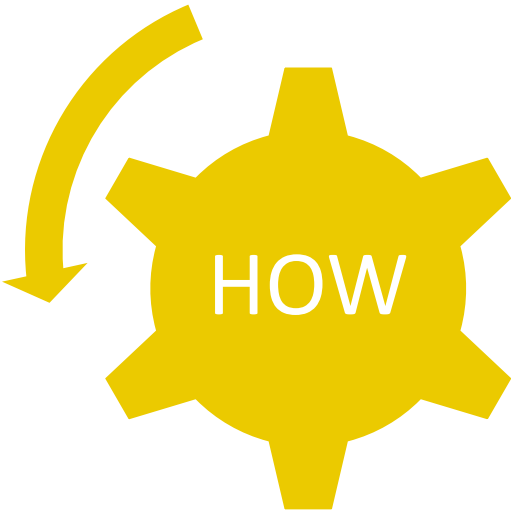
SVE strlen()

```
// ----------------------------------------
//      int strlen(const char *s) {
//          const char *e = s;
//          while (*e) e++;
//          return e - s;
//      }
// ----------------------------------------
// x0 = s

// Unoptimized SVE strlen
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
```


arm

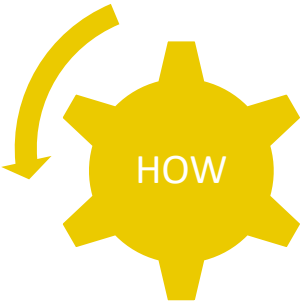
https://goo.gl/wZFwua



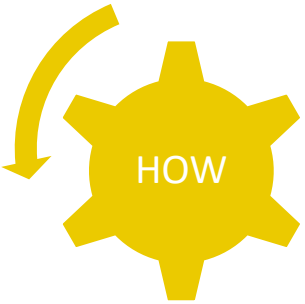
Vector Length Agnostic programming model

- Write once
- Compile once

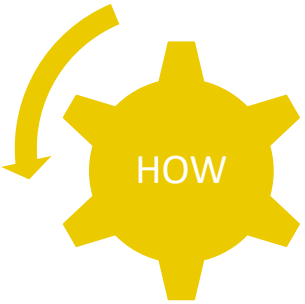
arm




```
float *a = /* ... */;
int N = /* ... */;
for (int i = 0; i < N; ++i)
  a[i] = 2.0 * a[i];
```

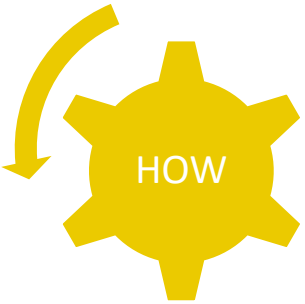



```
for (int i = 0; i < N; ++i)
  a[i] = 2.0 * a[i];
```




```
1 int i;
2 for (i = 0 ; (i < N - 3) && (N & ~3);
3       i += 4) {
4  float32x4_t va = vld1q_f32(&a[i]);
5  va = vmulq_n_f32(va, 2.0);
6  vst1q_f32(&a[i], va)
7 }
8 for (; i < N; ++i)
9   a[i] = 2.0 * a[i];
```


arm



Predication – get rid of the loop tail

```
for (int i = 0 ; i < N; i += 4) {
  svbool_t Pg = svwhilelt_b32(i, N);
  svfloat32_t va = svld1(Pg, &a[i]);
  va = svmul_x(Pg, va, 2.0);
  svst1(Pg, &a[i], va); }
```

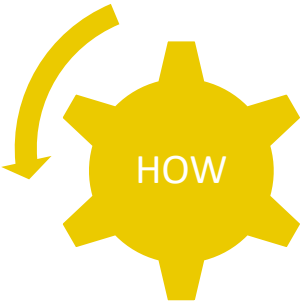
First vector iteration

3	2	1	0	Induction variable `i`
0x01	0x01	0x01	0x01	Pg

Second vector iteration

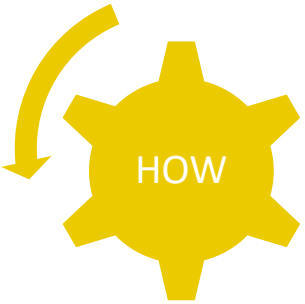
7	6	5	4	Induction variable `i`
0x00	0x01	0x01	0x01	Pg

arm



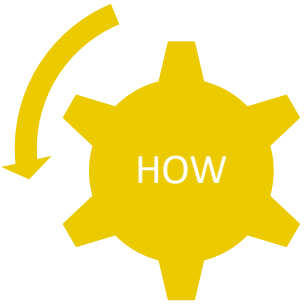
Go wider than 4 – how much?

```
for (int i = 0 ; i < N; i += 8) {
  svbool_t Pg = svwhilelt_b32(i, N);
  svfloat32_t va = svld1(Pg, &a[i]);
  va = svmul_x(Pg, va, 2.0);
  svst1(Pg, &a[i], va);
}
```



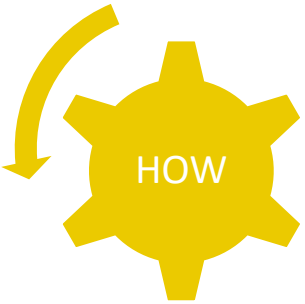
Go wider than 4 – how much?

```
for (int i = 0 ; i < N; i += 12) {
  svbool_t Pg = svwhilelt_b32(i, N);
  svfloat32_t va = svld1(Pg, &a[i]);
  va = svmul_x(Pg, va, 2.0);
  svst1(Pg, &a[i], va);
}
```



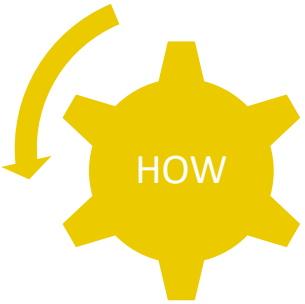
Go wider than 4 – how much?

```
for (int i = 0 ; i < N; i += 32) {
  svbool_t Pg = svwhilelt_b32(i, N);
  svfloat32_t va = svld1(Pg, &a[i]);
  va = svmul_x(Pg, va, 2.0);
  svst1(Pg, &a[i], va);
}
```



How about ANY width?

```
for (int i = 0 ; i < N; i += svcntw()) {
  svbool_t Pg = svwhilelt_b32(i, N);
  svfloat32_t va = svld1(Pg, &a[i]);
  va = svmul_x(Pg, va, 2.0);
  svst1(Pg, &a[i], va);
}
```



SVE Arm C Language Extension

```
for (int i = 0 ; i < N; i += svcntw()) {
  svbool_t Pg = svwhilelt_b32(i, N);
  svfloat32_t va = svld1(Pg, &a[i]);
  va = svmul_x(Pg, va, 2.0);
  svst1(Pg, &a[i], va);
}
```

C11: _Generic

svmul_x vmulq_n_f32

arm

Exercise

```
int argmax(float *arr, int n) {
  float maxVal = arr[0];
  int maxPos = 1;
  for (int i = 1; i < n; ++i) {
    if (arr[i] < maxVal) {
      maxVal = arr[i];
      maxPos = i;
    }
  }
  return maxPos;
}
```

arm

Thank You
Danke
Merci
谢谢
ありがとう
Gracias
Kiitos
감사합니다
धन्यवाद
تشكر