SC17 Arm SVE Workshop

Introducing the Armv8-A Scalable Vector Extension

Neil Burgess

Denver, CO

16 November 2017

Why is it called Scalable Vector Extension?

SVE is not a simple extension of AArch64 Advanced SIMD (“NEON”)

- A separate, optional architectural extension with a new set of instruction encodings.
- Initial focus is HPC and general-purpose server software, not media/image processing.

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.

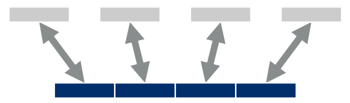
SVE begins to address traditional barriers to auto-vectorization

- 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)

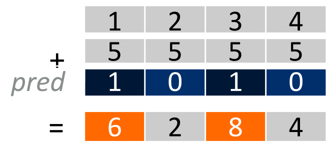
A vector extension to the Armv8-A architecture with some major new features

Gather-load and scatter-store



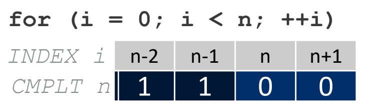
- Loads a single register from several non-contiguous memory locations.
- Enables vectorization of complex data structures with non-linear access patterns.

Per-lane predication



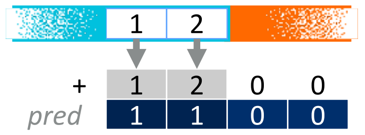
- Operations work on individual lanes under control of a predicate register.
- Enables vectorization of complex, nested control code containing side effects.

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.

New SVE architectural state

Scalable vector registers

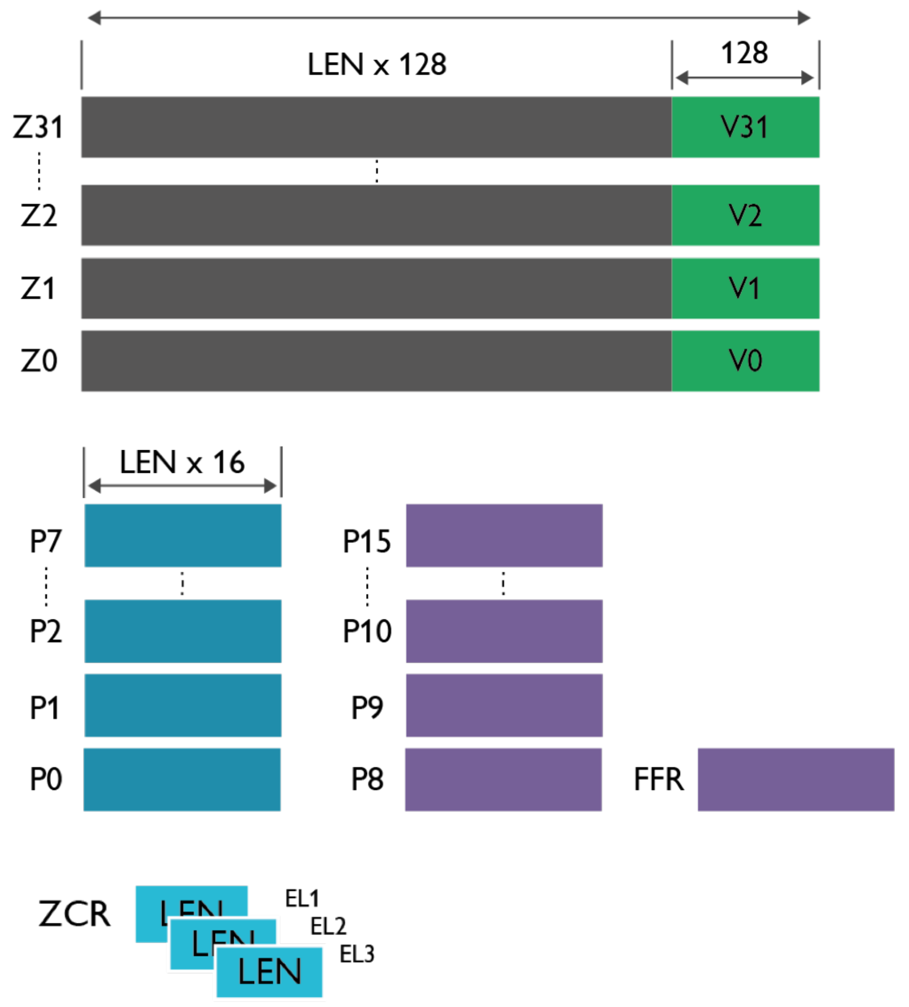
- Z0-Z31 extending NEON's V0-V31.
- Packed DP, SP & HP floating-point elements.
- Packed 64, 32, 16 & 8-bit integer elements.

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.



Predication

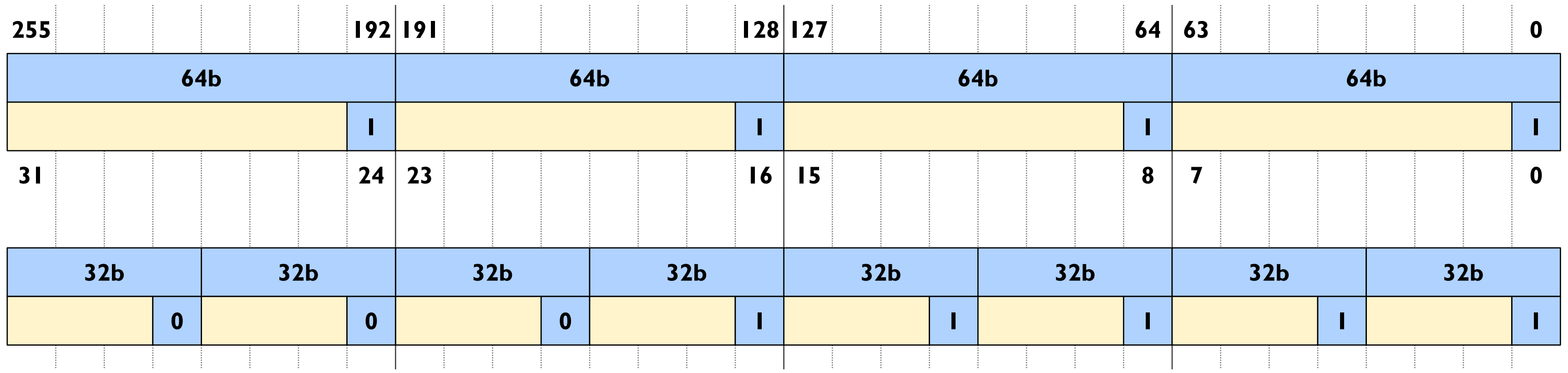
16 predicate registers defined (P0-P15)

- 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

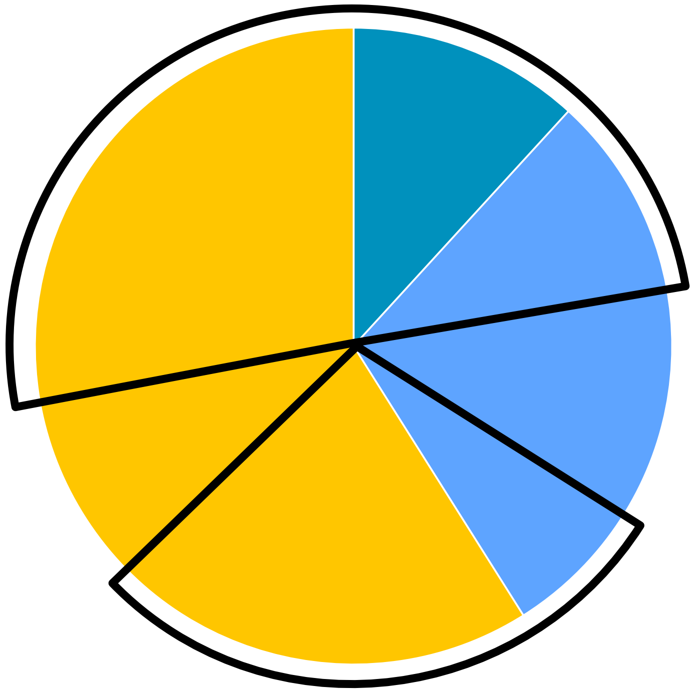
≈6% SVE instructions act on and return predicates → "Predicate ALU"



SVE instruction breakdown

1400 opcodes (cf 5000 AArch32 A32/T32; 4000 AArch64 A64; 1200 NEON)

- # opcodes includes element size but excludes immediates, regID's
  - SVE: A64 opcode[28:25] = 4'b0010
  - 130 basic operation "types"
  - 50% already in NEON
  - 30% "sort of" already in NEON



■ Float (12%)

■ Load/Store (29%)

■ Integer (59%)

SVE integer MUL/MAC operations

Instruction	src's & dst	Notes
MUL	Zds1, Zs2 / Pg -> Zds1, Zds1, imms2 -> Zds1	8-b signed immediate
MLA, MLS	Zs1, Zs2, Zds3 / Pg -> Zds3	overwrite addend
MAD, MSB	Zds1, Zs2, Zs3 / Pg -> Zds1	overwrite multiplicand
[SU]MULH	Zds1, Zs2 / Pg -> Zds1	16-b & 32-b only S**QD**MULH in NEON
INDEX	imms1, imms2 -> Zd imms1, Rs2 -> Zd Rs1, imms2 -> Zd Rs1, Rs2 -> Zd	Zd = s1 + (n × s2): n = 0 … #(elems - 1) Useful for constructing vector addresses 5-b unsigned immediates; 32-b or 64-b GPSR values

MAC: 48 opcodes, 4 "basic" types; none new

≈28 genuinely new types: LSU, PERM, vector and scalar ALU's, FPU

Differences between SVE and NEON

Load/Store

- Vectors used for addressing (gather/scatter)
- Predicated
- First-faulting behaviour
- Non-temporal
- Retain NEON “structured” load/stores

Speculative first-fault loads using FFR

Speculative gather from addresses in Z3

- **`SETFFR`**: initialises FFR to all TRUE
- A[2] is a Bad (e.g. unmapped) address

Initial state

P1	T	T	T	T
Z3	A[3]	Bad	A[1]	A[0]
FFR	T	T	T	T

SVE predicate condition flags

Overloads the ARM NZCV condition flags (reuse Aarch64 branch instructions)

More flag-setting instructions in SVE than NEON

- Vector compares set SVE flags

Logic for N and C identical to carry out from adder

Flag	SVE	Condition
N	First	Set if first active predicate element is true
Z	None	Set if no active predicate elements are true
C	!Last	Cleared if last active predicate element is true
V	n/a	Set to zero by most SVE instructions

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
```

More on Predication

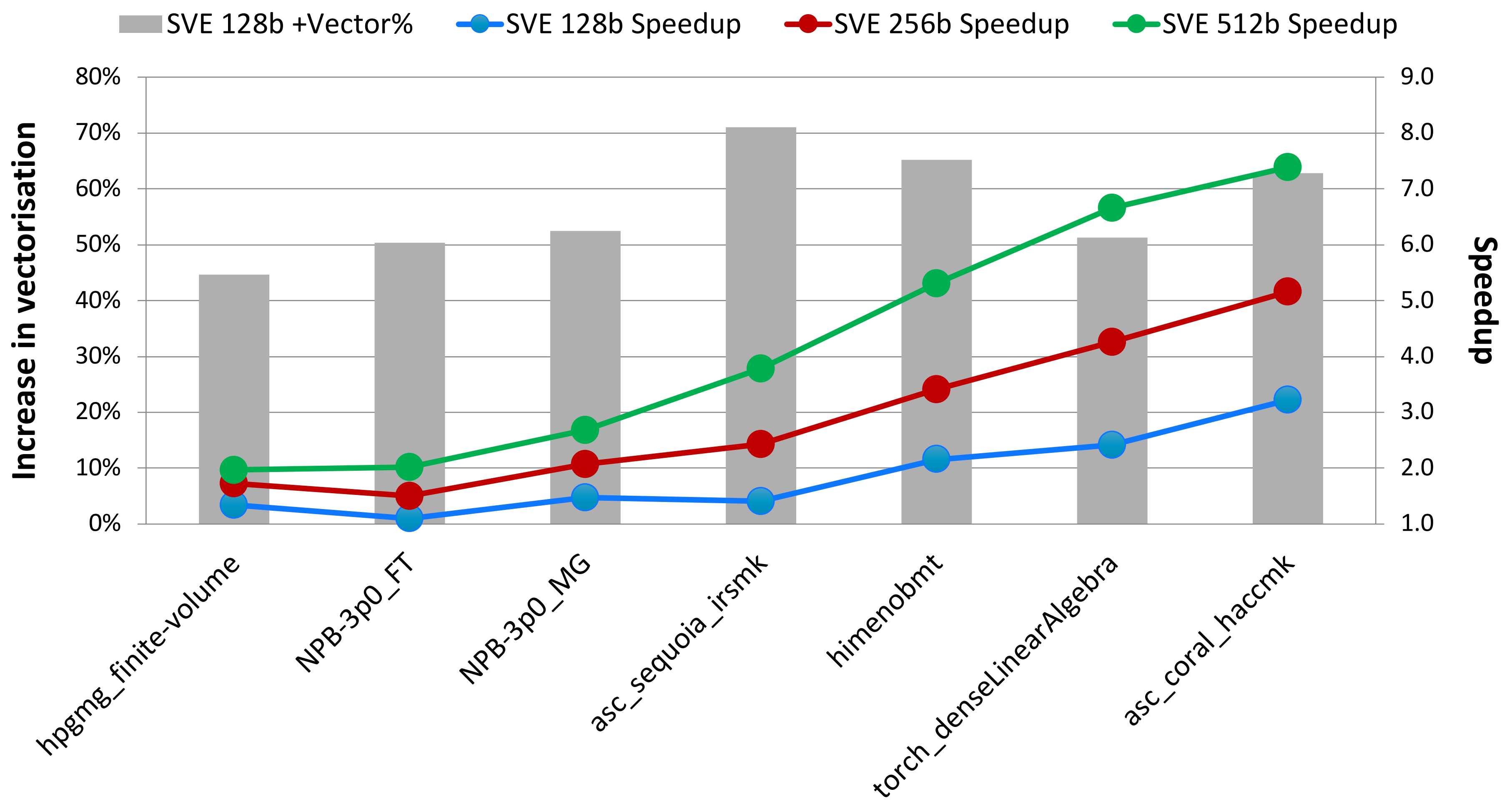
Other instructions that update predicates

- Vector compares return predicates
- WHILE instructions operate on two GPRs & return predicates
- Most predicate-generating instructions also write the NZCV condition flags
- Load/Store read predicates

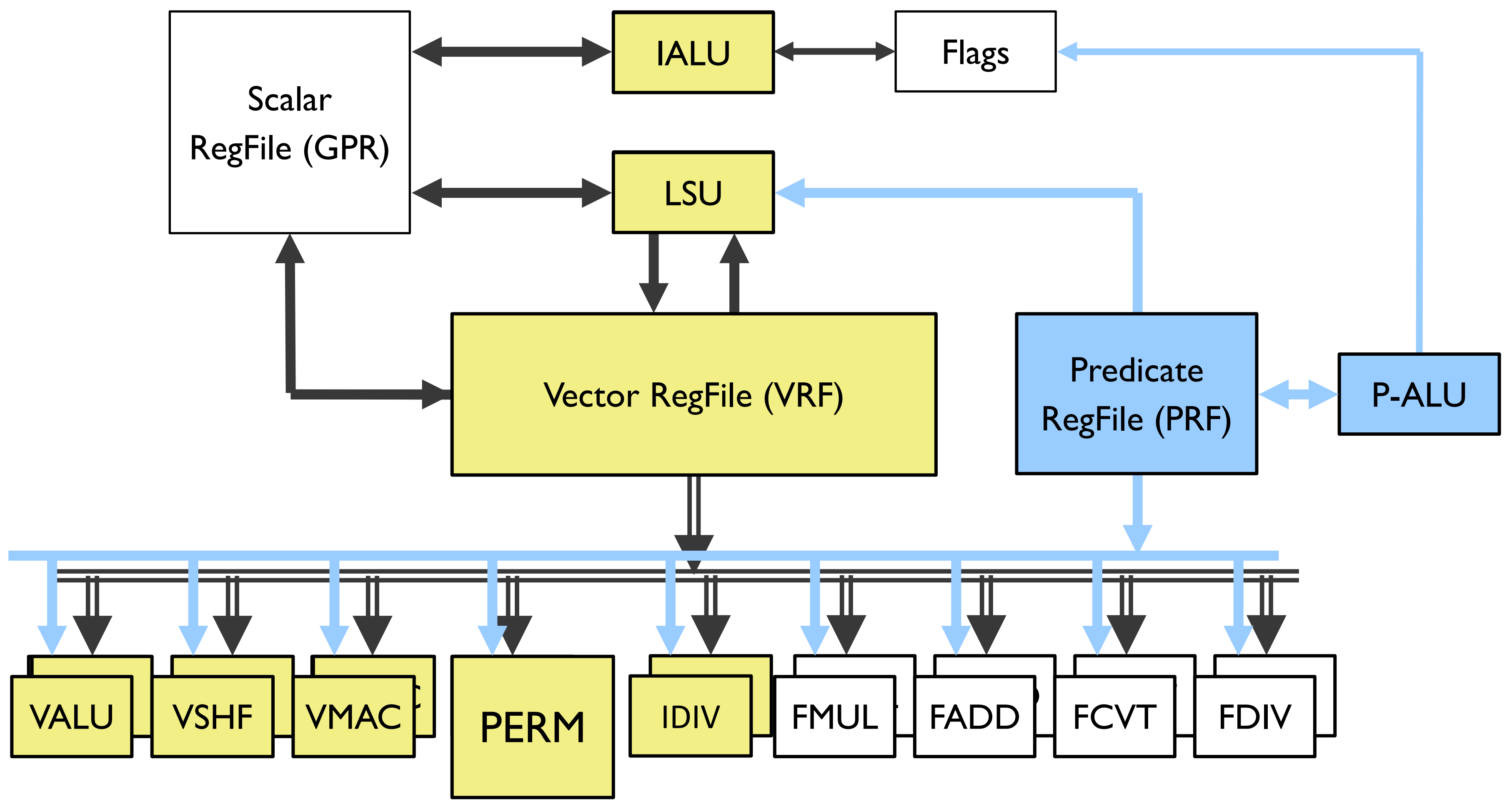
Hardware impact: predication mostly doesn’t affect execution

- can be implemented away from datapaths to share logic (muxes, operand staging)
- but sometimes it does e.g. vector reduction

HPC benchmarks SVE vs. NEON



SVE impact on CPU with NEON



SVE impact on area of vector execution units

Unit	Area Increase	New SVE functionality
VSHF	145%	ASRD, “wide” shfamt
VMAC	163%	64-b elements, INDEX
VALU	215%	“wide CMP”, NZCV logic, new ADR
PERM	418%	12 new “basic” operations

NEON 128-b vs SVE 256-b

- Double-size vectors → 100% area increase
- extra functionality & muxing; timing pressure

PERM block

- 256-b crossbar >5x area of 128-b
- Major increase in # op types executed

FPU area increase ≈100% (2x # units)

Overall execution area increase ≈ 2.2x

Overall CPU area increase ≈ 15%

- Decoder area increase of ≈4%
- Power increase broadly similar to area

Thank You!

Danke!

Merci!

谢谢!

ありがとう!

Gracias!

Kiitos!