Vector Architecture for HPC and ML

Alex Rico
Staff Research Engineer
Architecture Research

Severo Ochoa Research Seminars
BSC, UPC, July 2nd, 2018
Arm Architecture

Armv7 Advanced SIMD (aka Arm NEON instructions) now 12 years old

- Integer, fixed-point and non-IEEE single-precision float
- 16 × 128-bit vector registers

AArch64 Advanced SIMD was an evolution

- Gained full IEEE double-precision float and 64-bit integer vector ops
- 32 × 128-bit vector registers
Scalable Vector Extension – SVE

Significantly extends vector processing capabilities of AArch64

Enables implementation choices of vector lengths – **128 to 2048** bits

- *Vector Length Agnostic* (VLA) programming adjusts dynamically to the available VL
- No need to recompile, or to rewrite hand-coded SVE assembler or C intrinsics

Focus is HPC scientific workloads and machine learning, not media/image processing

Will enable advanced vectorizing compilers to extract more fine-grain parallelism from existing code and so reduce software deployment effort
Post-K Supercomputer goes Arm with SVE

Post-K Supports New SIMD Extension

- The SIMD extension is a 512-bit wide implementation of SVE
- SVE is an HPC-focused SIMD instruction extension in AArch64
  - Co-developed with ARM, taking advantage of Fujitsu’s HPC technologies
  - SVE and Advanced SIMD(NEON) are available, concurrently
- FUJITSU’s microarchitecture and compiler technologies maximize the execution performance of the Post-K CPU with SVE

Post-K CPU core
ARMv8-A with SVE
FUJITSU's microarchitecture

+ FUJITSU’s compiler

Good performance with practical applications
Post-K Prototype at ISC18

Major Specifications for Post-K

<table>
<thead>
<tr>
<th>Category</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction set architecture</td>
<td>Armv8-A SVE (512bit)</td>
</tr>
<tr>
<td>CPU</td>
<td></td>
</tr>
<tr>
<td>Number of cores</td>
<td>Computational nodes: 48 cores + 2 assistant cores</td>
</tr>
<tr>
<td></td>
<td>I/O and computational nodes: 48 cores + 4 assistant cores</td>
</tr>
<tr>
<td>Built-in interconnect</td>
<td>Tofu (6D Mesh/Torus)</td>
</tr>
<tr>
<td>System structure</td>
<td></td>
</tr>
<tr>
<td>Nodes</td>
<td>1 CPU/node</td>
</tr>
<tr>
<td>Racks</td>
<td>384 nodes/rack</td>
</tr>
<tr>
<td>Software</td>
<td></td>
</tr>
<tr>
<td>OS</td>
<td>Linux (RHEL-based) + McKernel (Lightweight Kernel)</td>
</tr>
<tr>
<td>System software</td>
<td>Successor to the Fujitsu Software Technical Computing Suite</td>
</tr>
<tr>
<td>Global file system</td>
<td>FEFS (Lustre-based)</td>
</tr>
<tr>
<td>Language</td>
<td>Successor to the Fujitsu Software Technical Computing Language (Fortran/C/C++, OpenMP, MPI), XcalableMP</td>
</tr>
<tr>
<td>Library framework</td>
<td>FDPS (Framework for Developing Particle Simulator)</td>
</tr>
</tbody>
</table>
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.

Per-lane predication
Operations work on individual lanes under control of a predicate register.

Predicate-driven loop control and management
Eliminate scalar loop heads and tails by processing partial vectors.
Introducing the Scalable Vector Extension (SVE)

A vector extension to the ARMv8-A architecture with some major new features:

Vector partitioning and software-managed speculation
First Faulting Load instructions allow memory accesses to cross into invalid pages.

Extended floating-point horizontal reductions
In-order and tree-based reductions trade-off performance and repeatability.
What’s the Vector Length?

There is no preferred vector length

- Vector Length (VL) is the CPU implementor’s choice, from 128 to 2048 bits, in increments of 128

- Adopting a Vector Length Agnostic (VLA) code generation style makes code portable across all possible vector lengths

- VLA is made possible by the per-lane predication, predicate-driven loop control, vector partitioning and software-managed speculation features of SVE

- No need to recompile, or to rewrite hand-coded SVE assembler or C intrinsics
SVE – Architectural State

- **Scalable vector registers**
  - Z0-Z31 extending NEON’s V0-V31
    - DP & SP floating-point
    - 64, 32, 16 & 8-bit integer
- **Scalable predicate registers**
  - P0-P7 lane masks for ld/st/arith
  - P8-P15 for predicate manipulation
  - FFR *first fault register*
- **Scalable vector control registers**
  - ZCR_ELx vector length (LEN=1..16)
  - Exception / privilege level EL1 to EL3
SVE Visual Examples
void daxpy(double *x, double *y, double a, int n)
{
    for (int i = 0; i < n; i++) {
        y[i] = a * x[i] + y[i];
    }
}

// x0 = &x[0]
// x1 = &y[0]
// x2 = &a
// x3 = &n

daxpy_:
    ldrsw x3, [x3]
    mov x4, #0
    ldr d0, [x2]
    b .latch
.loop:
    ldr d1, [x0, x4, lsl #3]
    ldr d2, [x1, x4, lsl #3]
    fmadd d2, d1, d0, d2
    str d2, [x1, x4, lsl #3]
    add x4, x4, #1
.latch:
    cmp x4, x3
    b.lt .loop
    ret
Loop fiberization: pulling multiple scalar iterations into a vector

**daxpy (SVE)**

```assembly
daxpy_
  ldrsw  x3, [x3]
  mov   x4, #0
  whilelt p0.d, x4, x3
  ld1rd z0.d, p0/z, [x2]
.loop:
  ld1d  z1.d, p0/z, [x0, x4, lsl #3]
  ld1d  z2.d, p0/z, [x1, x4, lsl #3]
  fmla z2.d, p0/m, z1.d, z0.d
  std z2.d, p0, [x1, x4, lsl #3]
  incd x4
.latch:
  whilelt p0.d, x4, x3
  b.first .loop
  ret
```

**daxpy (scalar)**

```assembly
daxpy_
  ldrsw  x3, [x3]
  mov x4, #0
  daxpy
  ldr  d0, [x2]
  .latch
  b .latch
.loop:
  ldr  d1, [x0, x4, lsl #3]
  ldr  d2, [x1, x4, lsl #3]
  fmadd  d2, d1, d0, d2
  str d2, [x1, x4, lsl #3]
  add x4, x4, #1
.latch:
  cmp x4, x3
  b.lt .loop
  ret
```

How do we handle the non-multiples of VL?
What happens at different vector lengths?
**daxpy (SVE – 128b)**

```
daxpy_
  ldrsw   x3, [x3]
  mov    x4, #0
  whilelt p0.d, x4, x3
  ld1rd   z0.d, p0/z, [x2]
.loop:
  ld1d    z1.d, p0/z, [x0,x4,lsl #3]
  ld1d    z2.d, p0/z, [x1,x4,lsl #3]
  fmla    z2.d, p0/m, z1.d, z0.d
  std1d   z2.d, p0, [x1,x4,lsl #3]
  incd    x4
  .latch:
  whilelt p0.d, x4, x3
  b.first .loop
  ret

256 ... 128 64

<table>
<thead>
<tr>
<th>Arrays</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>x[]</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>y[]</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```

Arrays

<table>
<thead>
<tr>
<th>256</th>
<th>128</th>
<th>64</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;x</td>
<td>&amp;y</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

CYCLES 2

© 2018 Arm Limited
daxpy (SVE – 128b)

```
daxpy_
  ldrsw  x3, [x3]
  mov   x4, #0
  whilelt p0.d, x4, x3
  ld1rd  z0.d, p0/z, [x2]
  .loop:
    ld1d  z1.d, p0/z, [x0,x4,ls1 #3]
    ld1d  z2.d, p0/z, [x1,x4,ls1 #3]
    fmla  z2.d, p0/m, z1.d, z0.d
    std1d z2.d, p0, [x1,x4,ls1 #3]
    incd  x4
  .latch:
    whilelt p0.d, x4, x3
    b.first .loop
    ret
```
daxpy (SVE – 128b)

```
daxpy_
   ldrsw  x3, [x3]
   mov   x4, #0
   whilelt p0.d, x4, x3
   ldlrd  z0.d, p0/z, [x2]
   .loop:
   ldl1d  z1.d, p0/z, [x0,x4,lsl #3]
   ldl1d  z2.d, p0/z, [x1,x4,lsl #3]
   fmla  z2.d, p0/m, z1.d, z0.d
   std1d  z2.d, p0, [x1,x4,lsl #3]
   incd  x4
   .latch:
   whilelt p0.d, x4, x3
   b.first .loop
   ret
```

Arrays
<table>
<thead>
<tr>
<th></th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>x[]</td>
<td>256</td>
<td>128</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>y[]</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Cycles: 4
daxpy (SVE – 128b)

daxpy_
    ldrsw x3, [x3]
    mov x4, #0
    whilelt p0.d, x4, x3
    ld1rd z0.d, p0/z, [x2]
    .loop:
        ld1d z1.d, p0/z, [x0,x4,lsl #3]
        ld1d z2.d, p0/z, [x1,x4,lsl #3]
        fmla z2.d, p0/m, z1.d, z0.d
        stdz z2.d, p0, [x1,x4,lsl #3]
    incd x4
    .latch:
        whilelt p0.d, x4, x3
        b.first .loop
        ret

<table>
<thead>
<tr>
<th>Arrays</th>
</tr>
</thead>
<tbody>
<tr>
<td>x[]</td>
</tr>
<tr>
<td>y[]</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CYCLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
</tr>
</tbody>
</table>
daxpy (SVE – 128b)

```
daxpy_:
    ldrsw x3, [x3]
    mov x4, #0
    whilelt p0.d, x4, x3
    ldlrd z0.d, p0/z, [x2]
    .loop:
      ldld z1.d, p0/z, [x0,x4,lsl #3]
      ldld z2.d, p0/z, [x1,x4,lsl #3]
      fmla z2.d, p0/m, z1.d, z0.d
      std ld z2.d, p0, [x1,x4,ls1 #3]
      incd x4
    .latch:
      whilelt p0.d, x4, x3
      b.first .loop
    ret
```

Arrays

<table>
<thead>
<tr>
<th></th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>x[]</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>y[]</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Arrays

<table>
<thead>
<tr>
<th></th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>x0</td>
<td>&amp;x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x1</td>
<td>&amp;y</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x3</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>T</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>T</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Cycles: 6
daxpy (SVE – 128b)

daxpy_
  ldrsw  x3, [x3]
  mov   x4, #0
  whilelt p0.d, x4, x3
  ld1rd  z0.d, p0/z, [x2]
  .loop:
    ld1d  z1.d, p0/z, [x0,x4,ls1 #3]
    ld1d  z2.d, p0/z, [x1,x4,ls1 #3]
    fmla  z2.d, p0/m, z1.d, z0.d
    stdz  z2.d, p0, [x1,x4,ls1 #3]
    incd  x4
  .latch:
    whilelt p0.d, x4, x3
    b.first .loop
    ret

&x
&y
3
0
T T

x0
x1
x3
x4

256  ...  128  64

arrays
<table>
<thead>
<tr>
<th>3 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>x[]</td>
</tr>
<tr>
<td>3 2 1 0</td>
</tr>
<tr>
<td>y[]</td>
</tr>
<tr>
<td>0 0 2 0</td>
</tr>
</tbody>
</table>

CYCLES 7

2.0 2.0
1.0 0.0
2.0 0.0
daxpy (SVE – 128b)

```
.daxpy_
  ldrsw  x3, [x3]
  mov    x4, #0
  whilelt p0.d, x4, x3
  ldlrd  z0.d, p0/z, [x2]
.loopt:
  ldld   z1.d, p0/z, [x0,x4,lsl #3]
  ldld   z2.d, p0/z, [x1,x4,lsl #3]
  fmla   z2.d, p0/m, z1.d, z0.d
  stld   z2.d, p0, [x1,x4,lsl #3]
  incd   x4
  .latcht:
    whilelt p0.d, x4, x3
    b.first .loop
    ret
```

<table>
<thead>
<tr>
<th></th>
<th>256</th>
<th></th>
<th></th>
<th>128</th>
<th>64</th>
</tr>
</thead>
<tbody>
<tr>
<td>x[]</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>y[]</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Arrays

<table>
<thead>
<tr>
<th>x[]</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>y[]</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

Arrays

<table>
<thead>
<tr>
<th>x0</th>
<th>x1</th>
<th>x3</th>
<th>x4</th>
<th>p0</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;x</td>
<td>&amp;y</td>
<td>3</td>
<td>3</td>
<td>T</td>
</tr>
</tbody>
</table>

Cycles

<table>
<thead>
<tr>
<th>z0</th>
<th>z1</th>
<th>z2</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.0</td>
<td>1.0</td>
<td>2.0</td>
</tr>
</tbody>
</table>

Cycles

<table>
<thead>
<tr>
<th></th>
<th>8</th>
</tr>
</thead>
</table>

© 2018 Arm Limited
**daxpy (SVE – 128b)**

```assembly
.daxpy_
    ldrsw x3, [x3]
    mov x4, #0
    whilelt p0.d, x4, x3
    ldld x0.d, p0/z, [x2]
    .loop:
        lda x1.d, p0/z, [x0,x4,lsl #3]
        lda x2.d, p0/z, [x1,x4,lsl #3]
        fmla x3.d, p0/m, x1.d, x2.d
        stld x4.d, p0, [x1,x4,lsl #3]
    .latch:
        whilelt p0.d, x4, x3
        b.first .loop
    ret
```

Arrays

<table>
<thead>
<tr>
<th></th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>x[]</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>y[]</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

- x0
- x1
- x3
- x4
- z0
- z1
- z2
- p0

CYCLES 9

Arrays

<table>
<thead>
<tr>
<th></th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>x[]</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>y[]</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>
daxpy (SVE – 128b)

daaxpy_

```
ldrsb x3, [x3]
mov x4, #0
whilelt p0.d, x4, x3
ld1rd z0.d, p0/z, [x2]
.loop:
  ld1d z1.d, p0/z, [x0,x4,lsl #3]
  ld1d z2.d, p0/z, [x1,x4,lsl #3]
  fmla z2.d, p0/m, z1.d, z0.d
  st1d z2.d, p0, [x1,x4,lsl #3]
  incd x4
.latch:
  whilelt p0.d, x4, x3
  b.first .loop
  ret
```

Arrays

<table>
<thead>
<tr>
<th></th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>x[]</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>y[]</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>x0</th>
<th>x1</th>
<th>x3</th>
<th>256</th>
<th>...</th>
<th>128</th>
<th>64</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;x</td>
<td>&amp;y</td>
<td>3</td>
<td>2</td>
<td>F</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>2.0</td>
<td>2.0</td>
<td>1.0</td>
<td>0.0</td>
<td>2.0</td>
<td>0.0</td>
<td></td>
</tr>
</tbody>
</table>

Cycles: 10
daxpy (SVE – 128b)

```assembly
.daxpy_

  ldrsw  x3, [x3]
  mov   x4, #0
  whilelt p0.d, x4, x3

  ld1rd  z0.d, p0/z, [x2]

  .loop:
    ldld  z1.d, p0/z, [x0,x4,lsl #3]
    ldld  z2.d, p0/z, [x1,x4,lsl #3]
    fmla  z2.d, p0/m, z1.d, z0.d
    stld  z2.d, p0, [x1,x4,lsl #3]
    incd  x4

  .latch:
    whilelt p0.d, x4, x3
  b.first .loop
  ret
```

Arrays

<table>
<thead>
<tr>
<th></th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>x[]</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>y[]</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

Arrays

<table>
<thead>
<tr>
<th></th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>x[]</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>y[]</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

256 ... 128 64

CYCLES 11
daxpy (SVE – 128b)

```assembly
.daxpy_
  ldrsw  x3, [x3]
  mov   x4, #0
  whilelt p0.d, x4, x3
  ldlrd z0.d, p0/z, [x2]
.loop:
  ldl1d z1.d, p0/z, [x0,x4,lsl #3]
  ldl1d z2.d, p0/z, [x1,x4,lsl #3]
  fmla z2.d, p0/m, z1.d, z0.d
  stld z2.d, p0, [x1,x4,1ls1 #3]
  incd x4
.latch:
  whilelt p0.d, x4, x3
  b.first .loop
  ret
```

<table>
<thead>
<tr>
<th>Array</th>
<th>256</th>
<th>...</th>
<th>128</th>
<th>64</th>
</tr>
</thead>
<tbody>
<tr>
<td>x[]</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>y[]</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

Arrays:
- Arrays: 3, 2, 1, 0
- x[]: 3, 2, 1, 0
- y[]: 0, 0, 2, 0

Cycles: 12
daxpy (SVE – 128b)

```assembly
.daxpy:
  ldrsw  x3, [x3]
  mov   x4, #0
  whilelt
  p0.d, x4, x3
  ld1rd z0.d, p0/z, [x2]
  .loop:
  ld1d  z1.d, p0/z, [x0,x4,lsl #3]
  ld1d  z2.d, p0/z, [x1,x4,lsl #3]
  fmla  z2.d, p0/m, z1.d, z0.d
  st1d  z2.d, p0, [x1,x4,lsl #3]
  incd  x4
  .latch:
  whilelt
  p0.d, x4, x3
  b.first .loop
  ret
```

<table>
<thead>
<tr>
<th>Arrays</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>x[]</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>y[]</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

Arrays:
- x[]: 3 2 1 0
- y[]: 0 0 2 0

Cycles: 13

256 128 64

&x &y

3 2

F T

2.0 2.0
0.0 2.0
0.0 4.0
**daxpy (SVE – 128b)**

```assembly
.daxpy:
  ldrsw x3, [x3]
  mov x4, #0
  whilelt p0.d, x4, x3
  ldlrd z0.d, p0/z, [x2]
  .loop:
    ld1d z1.d, p0/z, [x0,x4,lsl #3]
    ld1d z2.d, p0/z, [x1,x4,lsl #3]
    fmla z2.d, p0/m, z1.d, z0.d
    stld z2.d, p0, [x1,x4,ls1 #3]
  incd x4
  .latch:
    whilelt p0.d, x4, x3
    b.first .loop
  ret

<table>
<thead>
<tr>
<th>Arrays</th>
<th>64</th>
<th>128</th>
<th>256</th>
</tr>
</thead>
<tbody>
<tr>
<td>x[]</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>y[]</td>
<td>0</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>
```

Arrays:
- x: 0
- y: 4

Cycles: 14
daxpy (SVE – 128b)

```
daxpy_
   ldrsw  x3, [x3]
   mov   x4, #0
   whilelt p0.d, x4, x3
   ld1rd  z0.d, p0/z, [x2]
 .loop:
   ldld  z1.d, p0/z, [x0,x4,lsl #3]
   ldld  z2.d, p0/z, [x1,x4,lsl #3]
   fmla  z2.d, p0/m, z1.d, z0.d
   stld  z2.d, p0, [x1,x4,lsl #3]
   incd  x4
 .latch:
   whilelt p0.d, x4, x3
   b.first .loop
   ret
```

<table>
<thead>
<tr>
<th>Arrays</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>x[]</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>y[]</td>
<td>0</td>
<td>4</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

Arrays

<table>
<thead>
<tr>
<th>x[]</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>y[]</td>
<td>0</td>
<td>4</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

Cycles: 15

<table>
<thead>
<tr>
<th>&amp;x</th>
<th>&amp;y</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>T</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>x0</th>
<th>x1</th>
<th>x3</th>
<th>x4</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.0</td>
<td>2.0</td>
<td>0.0</td>
<td>2.0</td>
</tr>
<tr>
<td>0.0</td>
<td>2.0</td>
<td>0.0</td>
<td>4.0</td>
</tr>
</tbody>
</table>
daxpy (SVE – 128b)

daxpy_

ldrs w x3, [x3]
mov x4, #0
whilelt p0.d, x4, x3
ldldr z0.d, p0/z, [x2]

.loop:
ldld z1.d, p0/z, [x0,x4,lsl #3]
ldld z2.d, p0/z, [x1,x4,lsl #3]
fmla z2.d, p0/m, z1.d, z0.d
stld z2.d, p0, [x1,x4,lsl #3]
incd x4

.latch:
whilelt p0.d, x4, x3
b.first .loop
ret

Arrays

<table>
<thead>
<tr>
<th></th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>x[]</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>y[]</td>
<td>0</td>
<td>4</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

| 256 | ... | 128 | 64 |

<table>
<thead>
<tr>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>x0</td>
<td>x1</td>
<td>x3</td>
<td></td>
</tr>
<tr>
<td>x0</td>
<td>x1</td>
<td>x4</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2.0</th>
<th>2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>2.0</td>
</tr>
<tr>
<td>0.0</td>
<td>4.0</td>
</tr>
</tbody>
</table>

CYCLES 16

© 2018 Arm Limited
daxpy (SVE – 128b)

```assembly
    daxpy_
        ldrsw  x3, [x3]
        mov   x4, #0
        whilelt p0.d, x4, x3
        ldlrd  z0.d, p0/z, [x2]
    .loop:
        ldld  z1.d, p0/z, [x0,x4,ls1 #3]
        ldld  z2.d, p0/z, [x1,x4,ls1 #3]
        fmla  z2.d, p0/m, z1.d, z0.d
        stdld z2.d, p0, [x1,x4,ls1 #3]
        incd  x4
    .latch:
        whilelt p0.d, x4, x3
        b.first .loop
        ret
```

Arrays

<table>
<thead>
<tr>
<th></th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>x[]</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>y[]</td>
<td>0</td>
<td>4</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

Arrays

<table>
<thead>
<tr>
<th></th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>x[]</td>
<td>256</td>
<td>...</td>
<td>128</td>
<td>64</td>
</tr>
<tr>
<td>y[]</td>
<td>2.0</td>
<td>2.0</td>
<td>0.0</td>
<td>4.0</td>
</tr>
</tbody>
</table>

Cycles 17
daxpy (SVE – 128b)

daxpy_

    ldrsw  x3, [x3]
    mov   x4, #0
    whilelt p0.d, x4, x3
    ld1rd  z0.d, p0/z, [x2]

.loop:
    ld1d   z1.d, p0/z, [x0,x4,lsl #3]
    ld1d   z2.d, p0/z, [x1,x4,lsl #3]
    fmla   z2.d, p0/m, z1.d, z0.d
    st1d   z2.d, p0, [x1,x4,lsl #3]
    incd   x4

.latch:
    whilelt p0.d, x4, x3
    b.first .loop
    ret

<table>
<thead>
<tr>
<th>Arrays</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>x[]</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>y[]</td>
<td>0</td>
<td>4</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

256  ...  128  64

<table>
<thead>
<tr>
<th></th>
<th>2.0</th>
<th>2.0</th>
<th>2.0</th>
<th>4.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0</td>
<td>2.0</td>
<td>4.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

CYCLES  18
daxpy (SVE – 256b)

daxpy_:
    ldrsw x3, [x3]
    mov x4, #0
    whilelt p0.d, x4, x3
    ld1rd z0.d, p0/z, [x2]
    .loop:
        ld1d z1.d, p0/z, [x0,x4,lsl #3]
        ld1d z2.d, p0/z, [x1,x4,lsl #3]
        fmla z2.d, p0/m, z1.d, z0.d
        std1d z2.d, p0, [x1,x4,lsl #3]
        incd x4
    .latch:
        whilelt p0.d, x4, x3
        b.first .loop
        ret

<table>
<thead>
<tr>
<th>Arrays</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>x[]</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>y[]</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

256 ... 128 64

x0 &x
x1
x3
x4

3
0

F T T T T

z0
z1
z2

Cycles 2
daxpy (SVE – 256b)

```assembly
.daxpy_
  ldrsw x3, [x3]
  mov x4, #0
  whilelt p0.d, x4, x3
  ldlrd z0.d, p0/z, [x2]
  .loop:
    ldld z1.d, p0/z, [x0,x4,lsl #3]
    ldld z2.d, p0/z, [x1,x4,lsl #3]
    fmla z2.d, p0/m, z1.d, z0.d
    stdld z2.d, p0, [x1,x4,lsl #3]
    incd x4
  .latch:
    whilelt p0.d, x4, x3
    b.first .loop
  ret
```

<table>
<thead>
<tr>
<th>Arrays</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>x[]</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>y[]</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### Arrays

- `x[0-3]`: `&x`
- `y[0]`: `&y`
- `p0`: `F T T T`

### Cycles

- `0.0`: `z0`
- `2.0`: `z1`
- `2.0`: `z2`
- `2.0`: `z3`

---

Cycles: 3
daxpy (SVE – 256b)

```assembly
.daxpy_
    ldrsw  x3, [x3]
    mov   x4, #0
    whilelt p0.d, x4, x3
    ld1rd  z0.d, p0/z, [x2]
    .loop:
        ld1d  z1.d, p0/z, [x0,x4,lsl #3]
        ld1d  z2.d, p0/z, [x1,x4,lsl #3]
        fmla  z2.d, p0/m, z1.d, z0.d
        std1d z2.d, p0, [x1,x4,lsl #3]
        incd  x4
    .latch:
        whilelt p0.d, x4, x3
        b.first .loop
    ret
```

<table>
<thead>
<tr>
<th>Arrays</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>x[]</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>y[]</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

| CYCLES | 4 |

```
<table>
<thead>
<tr>
<th></th>
<th>x0</th>
<th>x1</th>
<th>x2</th>
<th>x3</th>
<th>x4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>256</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>128</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>64</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>F</th>
<th>T</th>
<th>T</th>
<th>T</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>z0</th>
<th>2.0</th>
<th>2.0</th>
<th>2.0</th>
<th>2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>z1</td>
<td>0.0</td>
<td>2.0</td>
<td>1.0</td>
<td>0.0</td>
</tr>
<tr>
<td>z2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

36 © 2018 Arm Limited
daxpy (SVE – 256b)

```
daxpy_
    ldrsw x3, [x3]
    mov x4, #0
    whilelt p0.d, x4, x3
    ldlrd z0.d, p0/z, [x2]
    .loop:
    ldlz z1.d, p0/z, [x0,x4,lsl #3]
    ldlz z2.d, p0/z, [x1,x4,lsl #3]
    fmla z2.d, p0/m, z1.d, z0.d
    stld z2.d, p0, [x1,x4,lsl #3]
    incd x4
    .latch:
    whilelt p0.d, x4, x3
    b.first .loop
    ret
```

Arrays

<table>
<thead>
<tr>
<th></th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>x[]</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>y[]</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Arrays

<table>
<thead>
<tr>
<th></th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>x0</td>
<td></td>
<td></td>
<td></td>
<td>&amp;x</td>
</tr>
<tr>
<td>x1</td>
<td></td>
<td></td>
<td></td>
<td>&amp;y</td>
</tr>
<tr>
<td>x3</td>
<td></td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>x4</td>
<td></td>
<td>4</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>p0</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>z0</td>
<td>0.0</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
</tr>
<tr>
<td>z1</td>
<td>0.0</td>
<td>2.0</td>
<td>1.0</td>
<td>0.0</td>
</tr>
<tr>
<td>z2</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

CYCLES 5
**daxpy (SVE – 256b)**

```assembly
.daxpy_
    ldrsw x3, [x3]
mov x4, #0
    whilelt p0.d, x4, x3
ld1rd z0.d, p0/z, [x2]
.loop:
    ld1d z1.d, p0/z, [x0,x4,1sl #3]
ld1d z2.d, p0/z, [x1,x4,1sl #3]
fmla z2.d, p0/m, z1.d, z0.d
std d z2.d, p0, [x1,x4,1sl #3]
incd x4
.latch:
    whilelt p0.d, x4, x3
b.first .loop
ret
```

<table>
<thead>
<tr>
<th>Arrays</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>x[]</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>y[]</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Arrays:
- x0 & x
- y & y

Cycles: 6

<table>
<thead>
<tr>
<th>Array</th>
<th>256</th>
<th>...</th>
<th>128</th>
<th>64</th>
</tr>
</thead>
<tbody>
<tr>
<td>x[]</td>
<td></td>
<td></td>
<td></td>
<td>x0</td>
</tr>
<tr>
<td>y[]</td>
<td></td>
<td></td>
<td></td>
<td>y0</td>
</tr>
</tbody>
</table>

- F: True
- T: True

<table>
<thead>
<tr>
<th>Array</th>
<th>0.0</th>
<th>2.0</th>
<th>2.0</th>
<th>2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>z0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>z1</td>
<td>0.0</td>
<td>2.0</td>
<td>1.0</td>
<td>0.0</td>
</tr>
<tr>
<td>z2</td>
<td>0.0</td>
<td>4.0</td>
<td>2.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

**Cycles:** 6
daxpy (SVE – 256b)

```assembly
.daxpy_
    ldrsw x3, [x3]
    mov x4, #0
    whilelt p0.d, x4, x3
    ldlrd z0.d, p0/z, [x2]
.loop:
    ldld z1.d, p0/z, [x0,x4,lsl #3]
    ldlld z2.d, p0/z, [x1,x4,lsl #3]
    fmla z2.d, p0/m, z1.d, z0.d
    stld z2.d, p0, [x1,x4,lsl #3]
    incd x4
.latch:
    whilelt p0.d, x4, x3
    b.first .loop
    ret
```

<table>
<thead>
<tr>
<th>Arrays</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>x[]</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>y[]</td>
<td>0</td>
<td>4</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CYCLES</th>
<th>7</th>
</tr>
</thead>
</table>
daxpy (SVE – 256b)

```assembly
.daxpy_
    ldrsw x3, [x3]
    mov x4, #0
    whilelt p0.d, x4, x3
    ld1rd z0.d, p0/z, [x2]
    .loop:
        ldld z1.d, p0/z, [x0,x4,lsl #3]
        ldld z2.d, p0/z, [x1,x4,lsl #3]
        fmla z2.d, p0/m, z1.d, z0.d
        stld z2.d, p0, [x1,x4,lsl #3]
        incd x4
    .latch:
        whilelt p0.d, x4, x3
        b.first .loop
    ret
```

<table>
<thead>
<tr>
<th>Arrays</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>x[0]</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>y[0]</td>
<td>0</td>
<td>4</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

256 ... 128 64

<table>
<thead>
<tr>
<th>Arrays</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>x[0]</td>
<td>&amp;x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y[0]</td>
<td>&amp;y</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

F T T T

<table>
<thead>
<tr>
<th>Arrays</th>
<th>0.0</th>
<th>2.0</th>
<th>2.0</th>
<th>2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>z[0]</td>
<td>0.0</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
</tr>
<tr>
<td>z[1]</td>
<td>0.0</td>
<td>2.0</td>
<td>1.0</td>
<td>0.0</td>
</tr>
<tr>
<td>z[2]</td>
<td>0.0</td>
<td>4.0</td>
<td>2.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

CYCLES 8
daxpy (SVE – 256b)

daxpy_

        ldrsw  x3, [x3]   
        mov   x4, #0
whilelt p0.d, x4, x3
        ldlrd z0.d, p0/z, [x2]
    .loop:
        ldld  z1.d, p0/z, [x0,x4,lsl #3]   
        ldld  z2.d, p0/z, [x1,x4,lsl #3]  
        FMLA z2.d, p0/m, z1.d, z0.d
        stld z2.d, p0, [x1,x4,lsl #3]
        incd x4
   .latch:
whilelt p0.d, x4, x3
b.first .loop
ret

<table>
<thead>
<tr>
<th>Cycl</th>
<th>256</th>
<th>128</th>
<th>64</th>
</tr>
</thead>
<tbody>
<tr>
<td>x[]</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>y[]</td>
<td>0</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

Arrays

<table>
<thead>
<tr>
<th>x0</th>
<th>x1</th>
<th>x2</th>
<th>x3</th>
</tr>
</thead>
<tbody>
<tr>
<td>z0</td>
<td>z1</td>
<td>z2</td>
<td>z4</td>
</tr>
</tbody>
</table>

Arrays 3 2 1 0

<table>
<thead>
<tr>
<th>&amp;x</th>
<th>&amp;y</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

x0 & x1 & y

<table>
<thead>
<tr>
<th>p0</th>
</tr>
</thead>
<tbody>
<tr>
<td>F  F F F</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>z0</th>
<th>z1</th>
<th>z2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>2.0</td>
<td>2.0</td>
</tr>
<tr>
<td>0.0</td>
<td>2.0</td>
<td>1.0</td>
</tr>
<tr>
<td>0.0</td>
<td>4.0</td>
<td>2.0</td>
</tr>
</tbody>
</table>

Cycles 9
**daxpy (SVE – 256b)**

```assembly
.daxpy_
    ldrsw  x3, [x3]
    mov   x4, #0
    whilelt p0.d, x4, x3
    ld1rd  z0.d, p0/z, [x2]
    .loop:
        ld1d  z1.d, p0/z, [x0,x4,lsl #3]
        ld1d  z2.d, p0/z, [x1,x4,lsl #3]
        fmla  z2.d, p0/m, z1.d, z0.d
        stld  z2.d, p0, [x1,x4,lsl #3]
        incd  x4
    .latch:
        whilelt p0.d, x4, x3
        b.first .loop
    ret
```

**Arrays**

<table>
<thead>
<tr>
<th></th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>x[]</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>y[]</td>
<td>0</td>
<td>4</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

**Arrays**

<table>
<thead>
<tr>
<th></th>
<th>x0</th>
<th>x1</th>
<th>x3</th>
</tr>
</thead>
<tbody>
<tr>
<td>z0</td>
<td>0.0</td>
<td>2.0</td>
<td>2.0</td>
</tr>
<tr>
<td>z1</td>
<td>0.0</td>
<td>2.0</td>
<td>1.0</td>
</tr>
<tr>
<td>z2</td>
<td>0.0</td>
<td>4.0</td>
<td>2.0</td>
</tr>
</tbody>
</table>

**Cycles**

10
daxpy (SVE – 256b)

daxpy_

        ldrsw  x3, [x3]
        mov   x4, #0
     whilelt p0.d, x4, x3
        ldlrd z0.d, p0/z, [x2]
      .loop:
          ldl d z1.d, p0/z, [x0,x4,ls1 #3]
          ldl d z2.d, p0/z, [x1,x4,ls1 #3]
          fmla z2.d, p0/m, z1.d, z0.d
          stld z2.d, p0, [x1,x4,ls1 #3]
           incd x4
     .latch:
          whilelt p0.d, x4, x3
            b.first .loop
            ret

<table>
<thead>
<tr>
<th>Arrays</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>x[]</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>y[]</td>
<td>0</td>
<td>4</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

256  ...  128  64

x0 &x

x1 &y

3 4

F  F  F  F

<table>
<thead>
<tr>
<th>Arrays</th>
<th>0.0</th>
<th>2.0</th>
<th>2.0</th>
<th>2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>x0</td>
<td>0.0</td>
<td>2.0</td>
<td>1.0</td>
<td>0.0</td>
</tr>
<tr>
<td>z0</td>
<td>0.0</td>
<td>4.0</td>
<td>2.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

CYCLES 11
Fault-tolerant Speculative Vectorization

Some loops have dynamic exit conditions that prevent vectorization

- E.g., the loop breaks on a particular value of the traversed array

<table>
<thead>
<tr>
<th>‘b’</th>
<th>‘s’</th>
<th>‘c’</th>
<th>‘-’</th>
<th>‘u’</th>
<th>‘p’</th>
<th>‘c’</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
</tr>
</tbody>
</table>

The access to unallocated space does not trap if it is not the first element

- Faulting elements are stored in the first-fault register (FFR)
- Subsequent instructions are predicated using the FFR information to operate only on successful element accesses
strlen (scalar)

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

```assembly
// x0 = s
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
```
### strlen (SVE)

```assembly
strlen:
    mov    x1, x0
    ptrue  p0.b

.loop:
    setffr
    ldfilb z0.b, p0/z, [x1]
    rdffr  p1.b, p0/z
    cmpeq  p2.b, p1/z, z0.b, #0
    brkbs  p2.b, p1/z, p2.b
    incp   x1, p2.b
    b.last .loop
    sub    x0, x1, x0
    ret
```

### strlen (scalar)

```assembly
// x0 = s
strlen:
    mov    x1, x0
    ptrue  p0.b

.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
```

Suboptimal implementation
strlen (SVE)

```
strlen:
    mov   x1, x0
    ptrue p0.b
.loop:
    setffr
    ldff1b z0.b, p0/z, [x1]
    rdffr p1.b, p0/z
    cmpeq p2.b, p1/z, z0.b, #0
    brkbs p2.b, p1/z, p2.b
    incp x1, p2.b
    b.last .loop
    sub   x0, x1, x0
    ret
```

Arrays

<table>
<thead>
<tr>
<th></th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>s[]</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>‘4’</td>
<td>‘ ’</td>
<td>‘m’</td>
<td>‘u’</td>
<td>‘t’</td>
<td>‘s’</td>
<td>‘o’</td>
<td>‘n’</td>
<td>‘e’</td>
<td>‘r’</td>
<td>‘a’</td>
<td>‘m’</td>
<td></td>
</tr>
</tbody>
</table>

CYCLES 0
strlen (SVE)

### strlen:

```
mov       x1, x0
ptrue    p0.b
.loop:
    setffr
    ldff1b  z0.b, p0/z, [x1]
    rdffr   p1.b, p0/z
    cmpeq   p2.b, p1/z, z0.b, #0
    brkbs   p2.b, p1/z, p2.b
    incp    x1, p2.b
    b.last  .loop
    sub     x0, x1, x0
    ret
```
**strlen (SVE)**

```
strlen:
  mov    x1, x0
  ptrue  p0.b
.loop:
  setffr
  ldff1b z0.b, p0/z, [x1]
  rdffr  p1.b, p0/z
  cmpeq  p2.b, p1/z, z0.b, #0
  brkbs  p2.b, p1/z, p2.b
  incp   x1, p2.b
  b.last .loop
  sub    x0, x1, x0
  ret
```

### Arrays

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>s[]</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>'4'</td>
<td>' '</td>
<td>'m'</td>
<td>'u'</td>
<td>'r'</td>
<td>'s'</td>
<td>'o'</td>
<td>'n'</td>
<td>'e'</td>
<td>'r'</td>
<td>'a'</td>
<td>'m'</td>
</tr>
</tbody>
</table>

### Memory Accesses

- **s[]**: 128 bytes
- **z0.b**: 128 bytes
- **p0.b**: 128 bytes
- **p1.b**: 128 bytes
- **p2.b**: 128 bytes

### Cycles

- Total Cycles: 2

---

© 2018 Arm Limited
**strlen (SVE)**

```assembly
cycles: 3

strlen:
    mov    x1, x0
    ptrue  p0.b
    .loop:
        setffr
        ldfllb z0.b, p0/z, [x1]
        rdffr  p1.b, p0/z
        cmpeq  p2.b, p1/z, z0.b, #0
        brkbs  p2.b, p1/z, p2.b
        incp   x1, p2.b
        b.last .loop
    sub    x0, x1, x0
    ret
```

Arrays: 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

| s[] | 0 0 0 '4' ' ' 'm' 'u' 'r' 's' 'o' 'n' 'e' 'a' 'm' |

Arrays: 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

128 ... 64 ... 32 16

| x0   | 0 |
| x1   | 0 |

FFR: F F T T T T T T T T T T T T T T T T

| p0   | T T T T T T T T T T T T T T T T T T |
| p1   | F F F F F F F F F F F F F F F F F F |
| p2   | F F F F F F F F F F F F F F F F F F |

| z0   | 0 0 0 4 |

... m u r t s o n e r a m
strlen (SVE)

strlen:
    mov    x1, x0
    ptrue  p0.b
    .loop:
    setffr
    ldfflb z0.b, p0/z, [x1]
    rdffr  p1.b, p0/z
    cmpeq  p2.b, p1/z, z0.b, #0
    brkbs  p2.b, p1/z, p2.b
    incp   x1, p2.b
    b.last .loop
    sub    x0, x1, x0
    ret

Arrays | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0
---------------------------
s[] | 0  | 0  | 0  | '4' | ' ' | 'm' | 'u' | 'r' | 's' | 'o' | 'n' | 'e' | 'r' | 'a' | 'm'

CYCLES  4
strlen (SVE)

Arrays

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>s[] 0</td>
<td>0</td>
<td>0</td>
<td>‘4’</td>
<td>‘ ’</td>
<td>‘m’</td>
<td>‘u’</td>
<td>‘r’</td>
<td>‘s’</td>
<td>‘o’</td>
<td>‘n’</td>
<td>‘e’</td>
<td>‘r’</td>
<td>‘a’</td>
<td>‘m’</td>
<td></td>
</tr>
</tbody>
</table>

strlen:

mov x1, x0
ptrue p0.b

.loop:

setffr
ldff1b z0.b, p0/z, [x1]
rdffr p1.b, p0/z
cmpeq p2.b, p1/z, z0.b, #0
brkbs p2.b, p1/z, p2.b
incp x1, p2.b
b.last .loop
sub x0, x1, x0
ret
strlen (SVE)

strlen:
  mov    x1, x0
  ptrue  p0.b
.loop:
  setffr z0.b, p0/z, [x1]
  lddf1b z0.b, p0/z, [x1]
  rddf r p1.b, p0/z
  cmpeq p2.b, p1/z, z0.b, #0
  brkbs p2.b, p1/z, p2.b
  incp  x1, p2.b
  b.last .loop
  sub    x0, x1, x0
  ret

Arrays

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>s[]</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>‘4’</td>
<td>‘ ’</td>
<td>‘m’</td>
<td>‘u’</td>
<td>‘t’</td>
<td>‘s’</td>
<td>‘o’</td>
<td>‘n’</td>
<td>‘e’</td>
<td>‘t’</td>
<td>‘a’</td>
<td>‘m’</td>
</tr>
</tbody>
</table>

128  ...  64  ...  32  16
x0
x1

FFR

p0

p1

p2

z0

arrays

s[]

CYCLES 6
strlen (SVE)

strlen:
    mov    x1, x0
  ptrue  p0.b
 .loop:
    setffr z0.b, p0/z, [x1]
    ldfflb z0.b, p0/z, [x1]
    rdfffr p1.b, p0/z
    cmpeq p2.b, p1/z, z0.b, #0
    brkbs p2.b, p1/z, p2.b
    incp  x1, p2.b
  b.last .loop
    sub    x0, x1, x0
    ret

Arrays
<table>
<thead>
<tr>
<th></th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>‘4’</td>
<td>‘ ’</td>
<td>‘m’</td>
<td>‘u’</td>
<td>‘r’</td>
<td>‘s’</td>
<td>‘o’</td>
<td>‘n’</td>
<td>‘e’</td>
<td>‘r’</td>
<td>‘a’</td>
<td>‘m’</td>
<td>---</td>
</tr>
</tbody>
</table>

Arrays
<table>
<thead>
<tr>
<th></th>
<th>128</th>
<th>...</th>
<th>64</th>
<th>...</th>
<th>32</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>x0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x1</td>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

FFR

p0

p1

p2

z0

s[]

Cycles

7
strlen (SVE)

Arrays

<table>
<thead>
<tr>
<th></th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>s[]</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>'4'</td>
<td>' '</td>
<td>'m'</td>
<td>'u'</td>
<td>'r'</td>
<td>'s'</td>
<td>'o'</td>
<td>'n'</td>
<td>'e'</td>
<td>'r'</td>
<td>'a'</td>
<td>'m'</td>
<td></td>
</tr>
</tbody>
</table>

```assembly
strlen:
  mov   x1, x0
  ptrue p0.b

.loop:
  setffr
  ldfflb z0.b, p0/z, [x1]
  rdffr p1.b, p0/z
  cmpeq p2.b, p1/z, z0.b, #0
  brkbs p2.b, p1/z, p2.b
  incp x1, p2.b
  b.last .loop
  sub   x0, x1, x0
  ret
```

Arrays

<table>
<thead>
<tr>
<th>128</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>x0</td>
<td></td>
</tr>
<tr>
<td>x1</td>
<td>13</td>
</tr>
</tbody>
</table>

FFR

```
F F T T T T T T T T T T T
```

p0

```
T T T T T T T T T T T T T
```

p1

```
F F T T T T T T T T T T T
```

p2

```
F F F T T T T T T T T T T
```

z0

```
0 0 0 4 m u r t s o n e r a m
```

CYCLES

8
strlen (SVE)

<table>
<thead>
<tr>
<th>Arrays</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>s[]</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>‘4’</td>
<td>‘ ’</td>
<td>‘m’</td>
<td>‘u’</td>
<td>‘i’</td>
<td>‘s’</td>
<td>‘o’</td>
<td>‘n’</td>
<td>‘e’</td>
<td>‘r’</td>
<td>‘a’</td>
<td>‘m’</td>
<td></td>
</tr>
</tbody>
</table>

```assembly
strlen:
  mov   x1, x0
  ptrue p0.b

.loop:
  setffr
  ldff1b z0.b, p0/z, [x1]
  rdffr p1.b, p0/z
  cmpeq p2.b, p1/z, z0.b, #0
  brkbs p2.b, p1/z, p2.b
  incp x1, p2.b
  b.last .loop
  sub   x0, x1, x0
  ret
```

---

CYCLES 9
strlen (SVE)

<table>
<thead>
<tr>
<th>Arrays</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>s[]</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>'4'</td>
<td>' '</td>
<td>'m'</td>
<td>'u'</td>
<td>'r'</td>
<td>'s'</td>
<td>'o'</td>
<td>'n'</td>
<td>'e'</td>
<td>'r'</td>
<td>'a'</td>
<td>'m'</td>
<td></td>
</tr>
</tbody>
</table>

```assembly
strlen:
    mov    x1, x0
    ptrue  p0.b

.loop:
    setffr z0.b, p0/z, [x1]
    ldff1b z0.b, p0/z
    rdffr  p1.b, p0/z
    cmpeq  p2.b, p1/z, z0.b, #0
    brkbs  p2.b, p1/z, p2.b
    incp   x1, p2.b
    b.last .loop
    sub    x0, x1, x0
    ret
```

<table>
<thead>
<tr>
<th>CYCLES</th>
<th>10</th>
</tr>
</thead>
</table>

Arrays:

- Arrays:
  - x0: 128
  - x1: 13

FFR:

- FFR: `FFFFTTTTTTTTTTT` (16)

Arrays:

- Arrays:
  - p0: `TTTTTTTTTTTTTTTTTTTT` (64)
  - p1: `FFFFTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT...
Gather-Load & Scatter-Store
Gather/Scatter Operations are Good and Evil

- Enable vectorization of codes with non-adjacent accesses on adjacent lanes
- Examples:
  - Outer loop vectorization
  - Strided accesses (larger than +1)
  - Random accesses
- Performance implementation dependent
  - Worst case one separate access per element
- LD1D <Zt>.D, Ps/Z [<Xn>, <Zm>.D]
Array of Structures vs. Structure of Arrays

typedef struct {
    uin64_t num_projects;
    float caffeine;
    bool cule_nmerengue;
} HPCProgrammer_t;

HPCProgrammer_t programmers[N];

typedef struct {
    uin64_t num_projects[N];
    float caffeine[N];
    bool cule_nmerengue[N];
} HPCProgrammer_t;

HPCProgrammer_t programmers;
Non-temporal Loads & Stores
SVE Non-Temporal Vector Instructions

- LDNT1D \{ <Zt1>.D \}, <Pglo>/Z, [<Xn|SP>, <Xm>, LSL #3]
- STNT1D \{ <Zt1>.D \}, <Pglo>, [<Xn|SP>{{, #<simm4>, MUL VL}}]

From the Arm ARM (Architecture Reference Manual):

Non-temporal contiguous load and stores include a hint to the memory system that this is a "streaming" access, and the memory locations are not expected to be accessed again soon so do not need to be retained in local caches.
Being Non-temporal is Not Enough

Vector Addition

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

No benefit if all accesses are temporal
Target to leave space for *temporal* accesses

```
ldnt1d
  c  +
ldnt1d
  b  =
stnt1d
  a
```
Mixed Temporal and Non-temporal
Mixed Temporal and Non-temporal

With non-temporal gather
And LRU allocation
Sparse Matrix Vector

for(m=row_start[j]; m<row_start[j+1]; m++)
    y[j] += A[m] * x[col[m]];

whilelt p1.d, xzr, x4
    ld1sw  z1.d, p1/z, [x2]   // z1 = &col[]
    ld1d   z2.d, p1/z, [x1, z1.d, lsl #3] // z2 = &x[col]\n    ld1d   z3.d, p1/z, [x0]   // z3 = &A[

    fm1a   z0.d, p1/m, z2.d, z3.d
    add    x2, x2, x7       // add half vector reg length (in bytes)
    addvl  x0, x0, 1       // add vector register length (in bytes)
    subs   x4, x4, x8       // Remaining length
    bgt    .L4
    faddv  d0, p0, z0.d     // result for y[j]
Vector Architecture
Design Trade-offs
Cache Coherent Vector Microarchitecture

Cores with one or more SVE and Load/Store (LS) unit(s)
Private L1 and L2 caches (considered part of the core)
System-level cache (SLC) shared among cores
Memory controllers (MC)
Network on chip (NoC) interconnects cores, SLCs and MCs

Disclaimer: Logical representation, not representative of a physical implementation
SVE Execution Pipeline

Vector length
- Vectorized code will execute less instructions
- Vector register file size

Number of execution units and width
- Determines computation throughput

Vector instruction latencies, cracking, etc...

Example:
- Core implements SVE-256 – registers are 256-bit wide
  - There are two execution units of 256 bits (dual issue)
  - Peak throughput per core is 512b/cycle
- A smaller core could implement SVE-256 but one 128b exec unit
  - Each instruction would use two issue cycles
  - Peak throughput per core would be 128b/cycle
Load-Store Execution Pipeline

Number of Load-Store execution units
L1 maximum access width
L1 concurrent accesses
  • Number of ports
  • Number of banks/arrays
L1 cache size
Prefetching aggressiveness
L2 Cache

L2 size to filter NoC accesses
Prefetching aggressiveness

Core

L2
L1
SVE
LS

NoC

SLC
SLC
MC
MC
Network on Chip

Bandwidth
Connectivity – topology, number of links
Routing to reduce congestion
System Level Cache

SLC size, prefetching, replacement to filter main memory accesses
Memory

Memory bandwidth

- Channels, banks, width,...

HBM vs DRAM vs NVM...
Memory hierarchy and NoC to feed data to SVE units

SVE unit configuration for target throughput
SVE Programming and Tools
SVE Programming

Assembly

Full ISA Specification:
The Scalable Vector Extension for Armv8-A

Lots of worked examples in A sneak peek into SVE and VLA programming

Intrinsics

Arm C Language Extensions for SVE
Arm Scalable Vector Extensions and application to Machine Learning

 Compiler

Autovectorization – GCC, Arm Compiler for HPC, Cray, Fujitsu
Help the compiler: OpenMP

#pragma omp parallel for simd
SVE Tools

Arm Compiler

Arm Instruction Emulator

Research Enablement Kit

Arm v8-A Binary

Arm v8-A Binary with new features

Arm Instruction Emulator

Linux

Arm v8-A
Comprehensive C/C++/Fortran compiler with best-in-class performance

Tuned for Scientific Computing, HPC and Enterprise workloads
- Processor-specific optimizations for various server-class Arm-based platforms
- Optimal shared-memory parallelism using latest Arm-optimized OpenMP runtime

Linux user-space compiler with latest features
- C++ 14 and Fortran 2003 language support with OpenMP 4.5*
- Support for Armv8-A and SVE architecture extension
- Based on LLVM and Flang, leading open-source compiler projects

Commercially supported by Arm
- Available for a wide range of Arm-based platforms running leading Linux distributions – RedHat, SUSE and Ubuntu
Arm Compiler – Building on LLVM, Clang and Flang projects

Arm C/C++/Fortran Compiler

- C/C++ Files (.c/.cpp)
- Fortran Files (.f/.f90)
- C/C++ Frontend (Clang based)
- Fortran Frontend (PGI Flang based)
- LLVM IR
- IR Optimizations
  - Auto-vectorization
  - Enhanced optimization for Armv8-A and SVE
- Language specific frontend
- Language agnostic optimization
- Architecture specific backend
- LLVM based
- Armv8-A code-gen
- LLVM IR
- SVE code-gen
- LLVM based
- Armv8-A binary
- SVE binary
Arm Instruction Emulator

Develop your user-space applications for future hardware today

Start porting and tuning for future architectures early
- Reduce time to market, Save development and debug time with Arm support

Run 64-bit user-space Linux code that uses new hardware features on current Arm hardware
- SVE support available now. Support for 8.x planned.
- Tested with Arm Architecture Verification Suite (AVS)

Near native speed with commercial support
- Emulates only unsupported instructions
- Integrated with other commercial Arm tools including compiler and profiler
- Maintained and supported by Arm for a wide range of Arm-based SoCs

© 2018 Arm Limited
Arm Instruction Emulator

Develop your user-space applications for future hardware today

Run Linux user-space code that uses new hardware features (SVE) on current Arm hardware

Simple “black box” command line tool

$ armclang hello.c --march=armv8+sve
$ ./a.out
Illegal instruction
$ armie -msve-vector-bits=256 ./a.out
Hello

Converts unsupported instructions to native Armv8-A instructions
DynamoRIO Instruction Count

- Counts AArch64 Instructions and specifically SVE instructions
- SVE instructions are recorded in binary.sve.instrs
- decode utility can be used to make the traces human-readable

```
armie -c inscount -e -msve-vector-bits=256 ./sve_binary
cat sve_binary.sve.instrs \  
  | awk '{printf("%s %d\n",$2,$3)}\'  
  | while read x c; do decode $x $c; done
```
Instruction Count - ZGEMM example

Left: reference FORTRAN implementation, right: custom kernel.
Arm Research Enablement Kit – System Modeling Using gem5

https://developer.arm.com/research/research-enablement/system-modeling

- HPI: First Armv8-A based CPU timing model released by Arm
- Documentation about the HPI core model (based on MinorCPU)
- Documentation about running benchmarks (PARSEC)
- Useful scripts (clone.sh, read_results.sh)
  - Using the current mainline gem5 source code
  - SVE patches will be upstreamed after completing beta testing
More on SVE

- Intrinsics: [Arm C Language Extensions for SVE](http://developer.arm.com/hpc)
- Lots of worked examples in [A sneak peek into SVE and VLA programming](http://developer.arm.com/hpc)
- Optimized machine learning in [Arm SVE and application to Machine Learning](http://developer.arm.com/hpc)
Future SVE
Contact me for opportunities

Internships 2019 (check September-October)

• Sophia Antipolis, France
• Cambridge, UK

Full-time positions

arm.com/careers
Research Summit
17-19 September 2018
Robinson College, Cambridge, UK

• Architecture & Memory
• Biotechnology
• Flexible Electronics
• High Performance Computing
• Internet of Things
• Large Scale Systems
• Machine Learning
• Security & Servers

With more still to be announced!

arm.com/summit
The Arm trademarks featured in this presentation are registered trademarks or trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. All rights reserved. All other marks featured may be trademarks of their respective owners.

www.arm.com/company/policies/trademarks
Thank You!
Danke!
Merci!
谢谢!
ありがとう!
Gracias!
Kiitos!