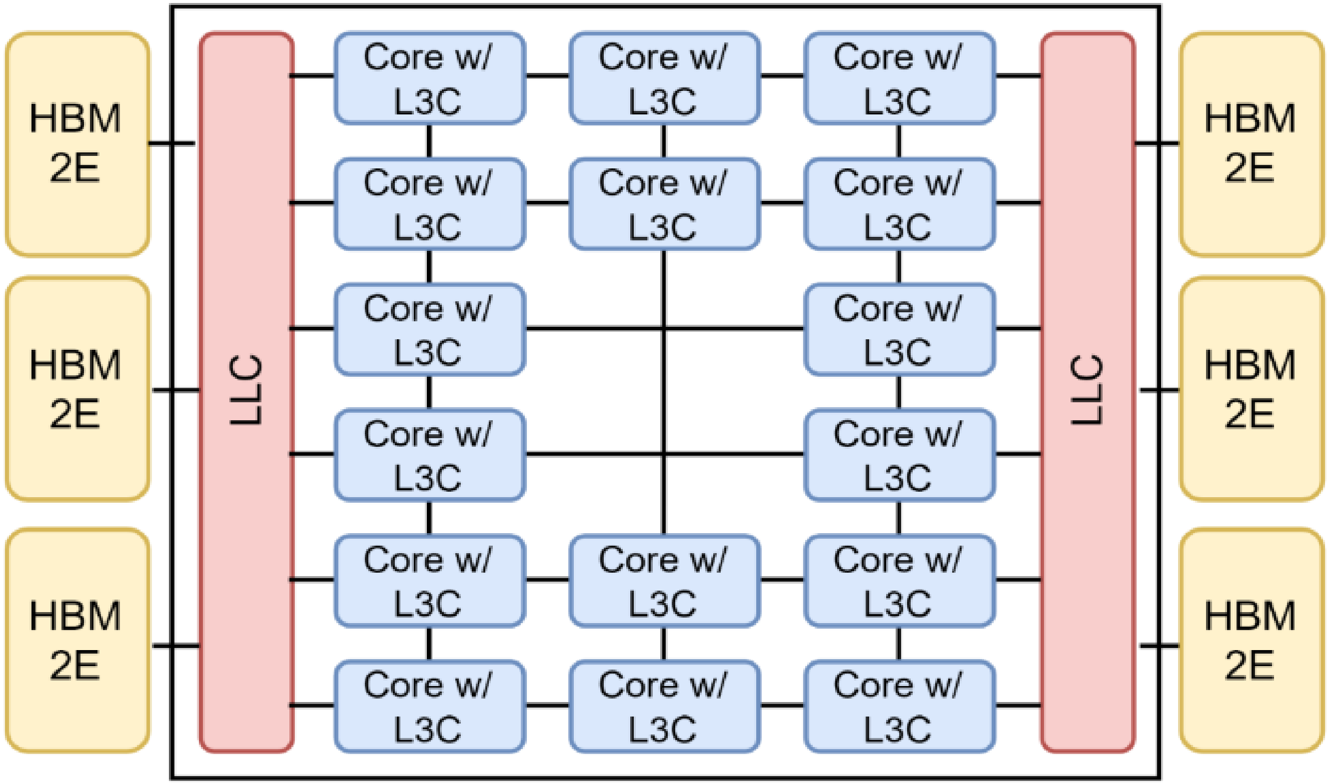
RISC-V Vector versus SX-Aurora

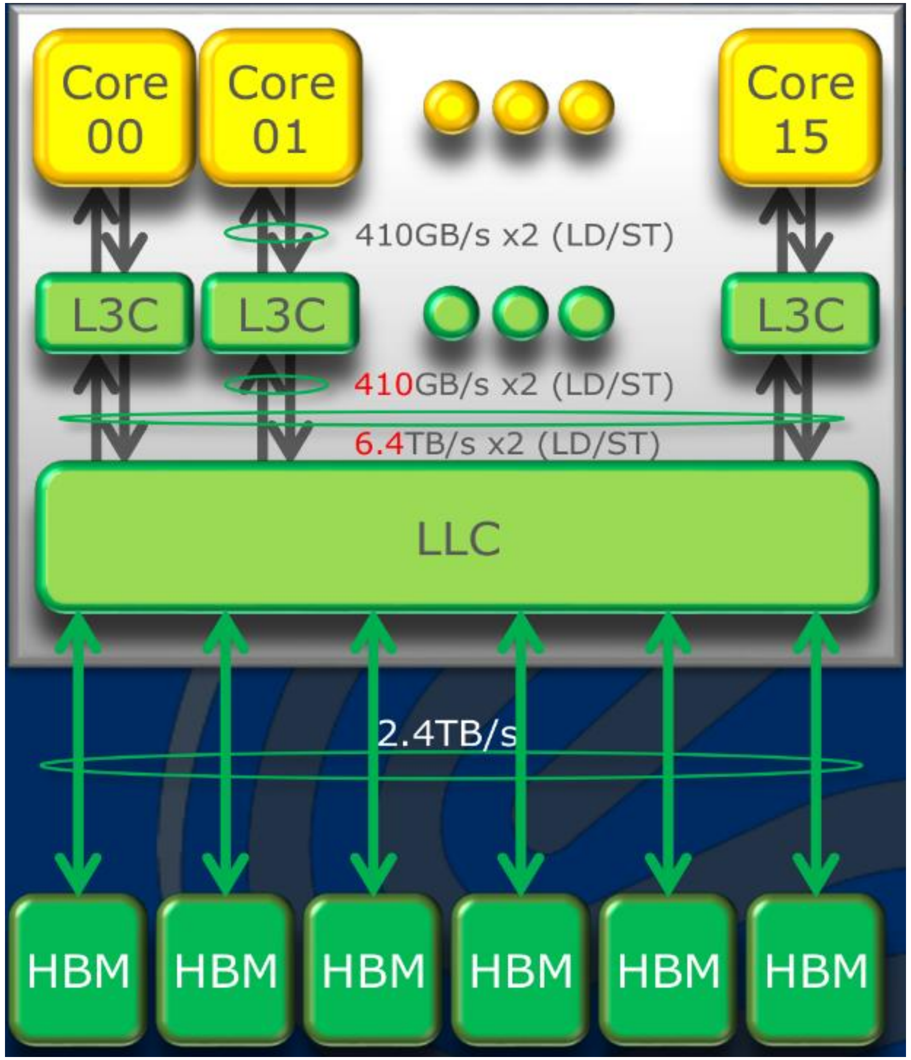
Dr. Erich Focht
HPC Fellow
OpenChip & Software Technologies

NEC User Group, Hamburg
June 12-14, 2024

NEC SX-AURORA TSUBASA

VE3 ARCHITECTURE





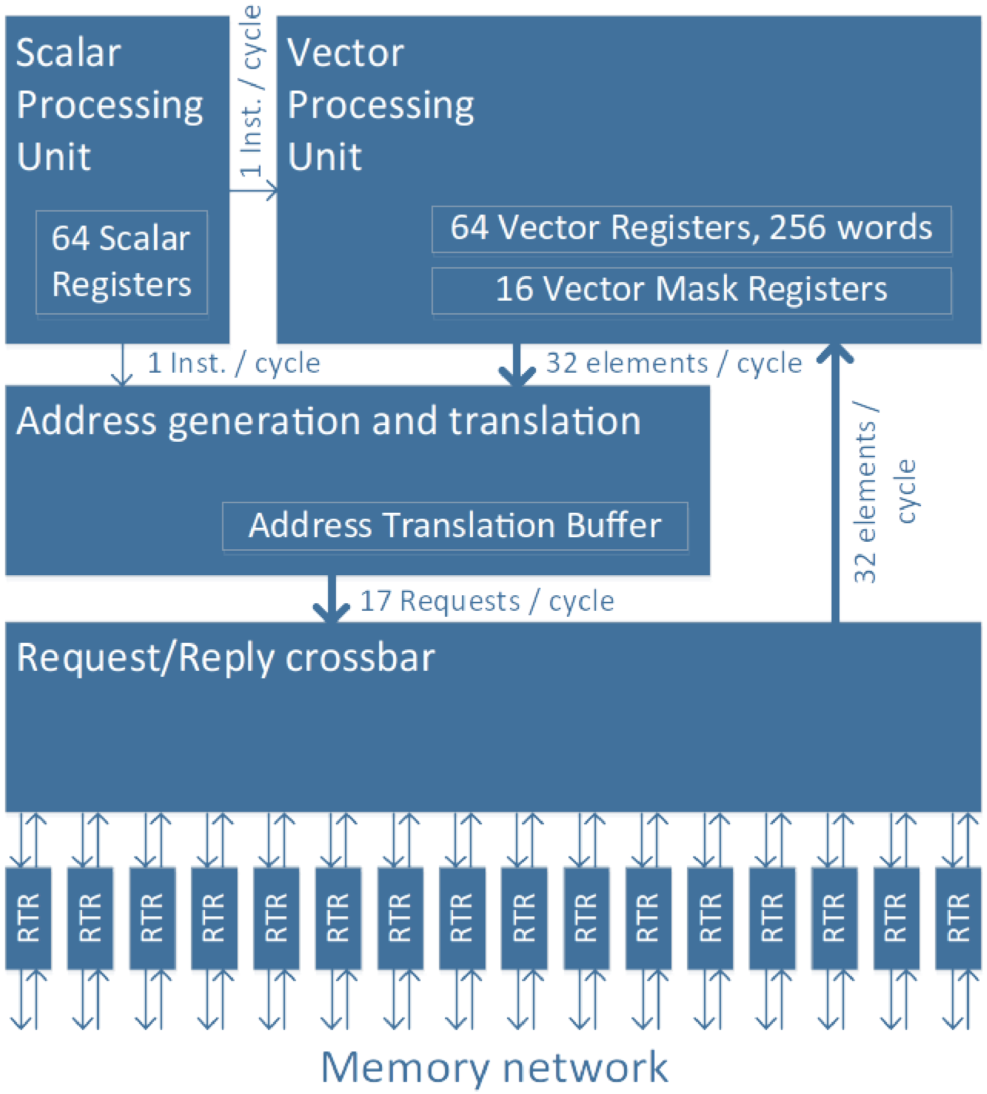
NEC SX-AURORA TSUBASA

Enabling the Digital World

VE3 Specs

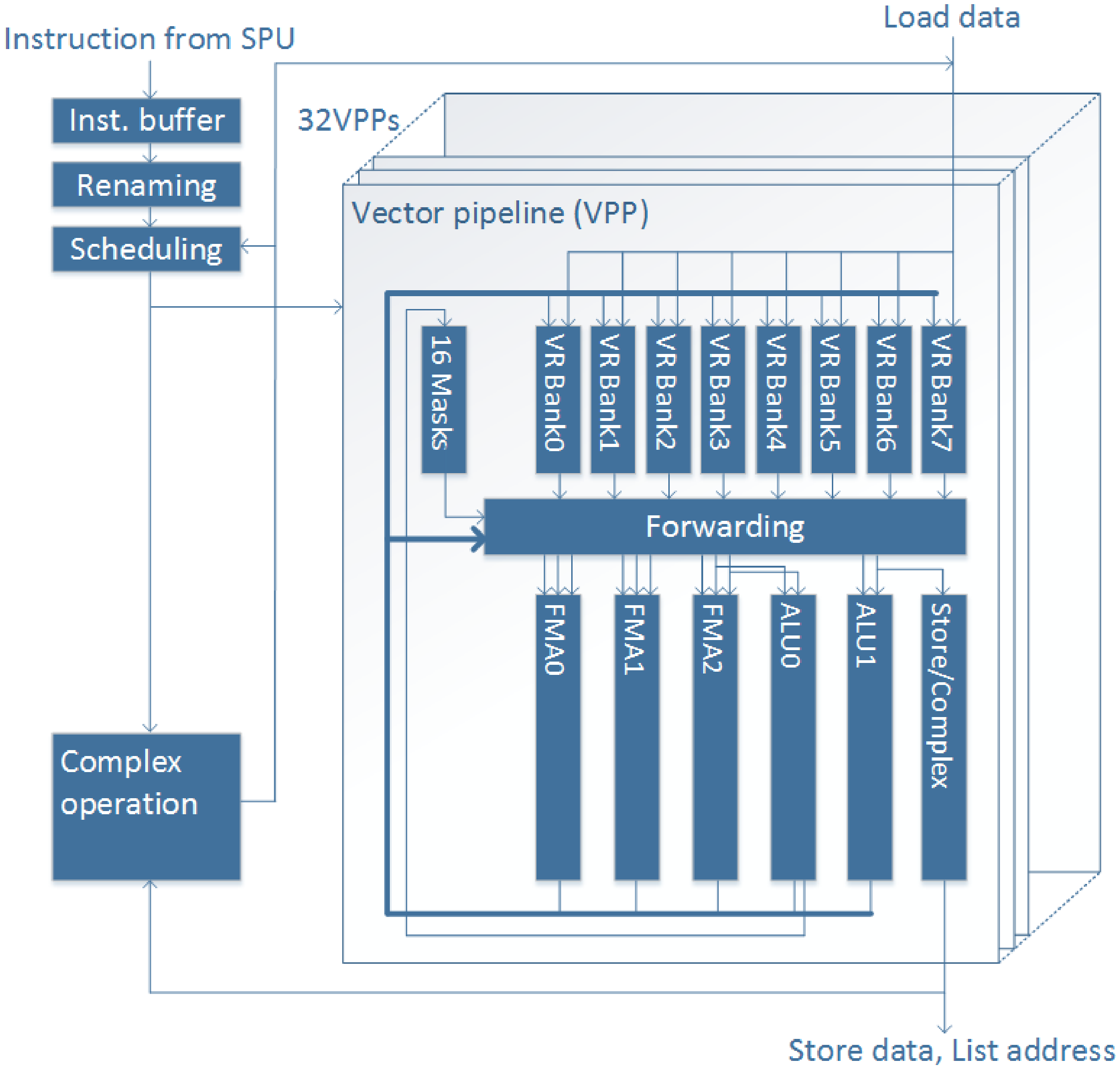
cores/CPU	16
core performance	307GF(DP) 614GF(SP)
CPU performance	4.9TF(DP) 9.8TF(SP)
L3 cache	2MB per core SW controllable
LL cache	64MB shared
memory bandwidth	2.4TB/s
Memory capacity	96GB
PCIe	Gen4

VE1/2 Core



NEC SX-AURORA TSUBASA

openchip
Enabling the Digital World



VPU: Vector Processing Unit

- Out of order
- VREG renaming
- 32 Vector Pipelines / Lanes each with:
  - 3 x FMAs
  - 2 x ALUs
  - 1 x Store / Complex unit
  - 8 banks VREG file
  - 16 vector masks
  - Forwarding stage for vector instruction chaining and complex operations (DIV , SQRT)

NEC SX-AURORA: VECTOR ISA

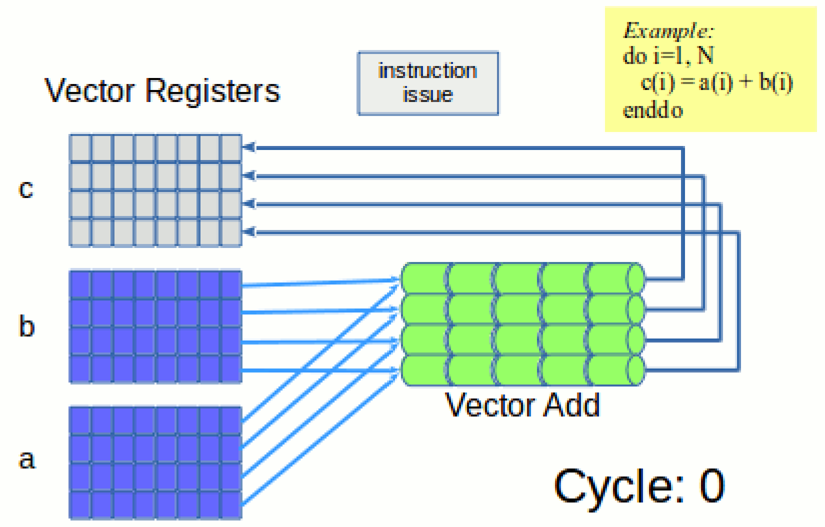
Data types: fp64, fp32, (u)int64, (u)int32

64 Vector Registers (VREG)
- Vector Register Size is defined by architecture
- 256 x 64bit = 16384 bits
- 512 x 32bit (packed)

16 Vector Mask Registers (VM)
- 256 bits
- Need two consecutive masks for packed

1 Vector Length Register (VL)
- Operate only on first VL elements of the VREG
- Valid for all vector ops until VL is changed

Vector Operations (...)
- No precise exceptions, not restartable
- Don't tolerate page or TLB faults.



RISC-V ISA

RISC-V is an open, free ISA

- No license required to implement it
- Created at UC Berkeley in 2010

Development within RISC-V International

- Technical resources, learning
- Committees and Working Groups
- Specifications and software tools

Base RISC-V ISA is simple and minimalistic

- MIPS-alike, either 32 or 64 bit integers (RV32I, RV64I)
- ~50 instructions, only integer arithmetic
- Load/store based architecture, one addressing mode

Premier

Premier TSC

chip

g the Digital World

SC-V

Strategic

landscape.riscv.org

openchip
Enabling the Digital World

RISC-V ISA EXTENSIONS

RISC-V Extensions

- RISC-V is augmented via the concept of extensions
  - Extensions can add new instructions and CPU state
- Base ISA is called I (for Integer)
  - RV32I
  - RV64I (XLEN=64, adds a few arithmetic instructions to improve 32-bit integer arithmetic)
- Common Standard Extensions in a RISC-V 64-bit Linux capable core

IMAFD = G

  - M. Integer multiplication and division (mul, div, rem, ...)
  - A. Atomic instructions (load reserve + store conditional, atomic read-modify-write)
  - F. Single-Precision Floating-Point (IEEE 754 Binary32)
  - D. Double-Precision Floating-Point (IEEE 754 Binary64)
  - C. Compressed Instructions (16-bit encodings for common I/F/D instructions)

BSC Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Roger Ferrer Ibanez

RISC-V ISA VECTOR EXTENSION

RISC-V decided for a VECTOR ISA

- Not a SIMD ISA!
- Variable vector length
  - Vector length register like SX Aurora
- Vector register size not prescribed
  - VLEN = VREG size in bits, **is not fixed!**
  - Left to the implementation
  - Must be power of 2
  - ELEN ≥ 8 max element size in bits
  - ELEN ≤ VLEN ≤ 65536

Code can be VLEN agnostic! Runs on any impl.

https://github.com/riscv/riscv-v-spec

RISC-V ISA VECTOR EXTENSION

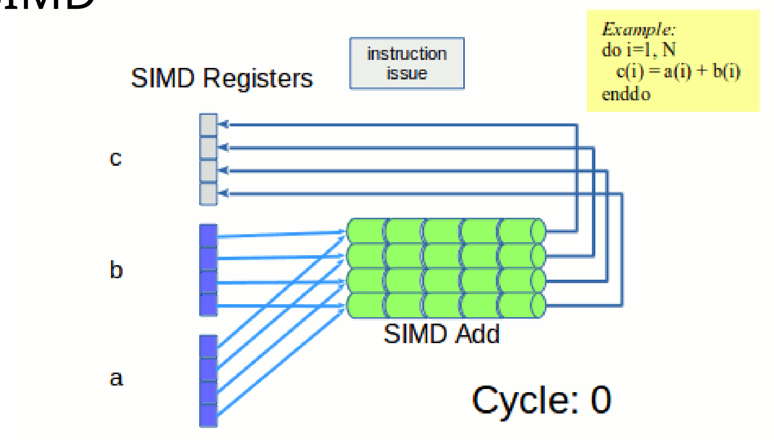
RISC-V decided for a VECTOR ISA

- Not a SIMD ISA!
- Variable vector length
  - Vector length register like SX Aurora
- Vector register size not prescribed
  - VLEN = VREG size in bits, **is not fixed!**
  - Left to the implementation
  - Must be power of 2
  - ELEN ≥ 8 max element size in bits
  - ELEN ≤ VLEN ≤ 65536

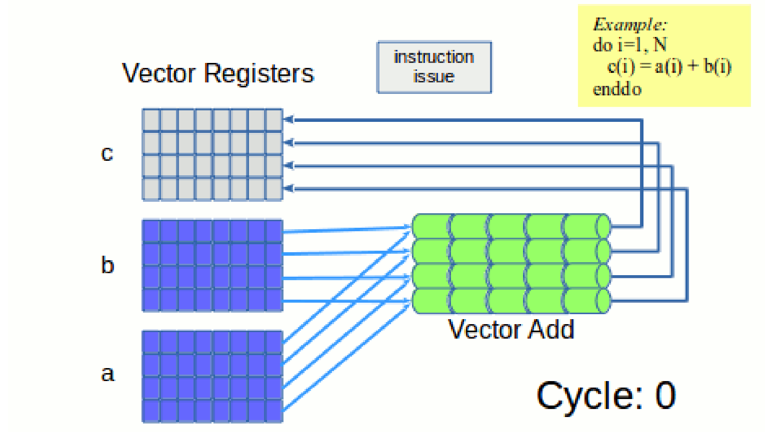
Code can be VLEN agnostic! Runs on any impl.

https://github.com/riscv/riscv-v-spec

SIMD



SIMD+pipeline (temporal vector)



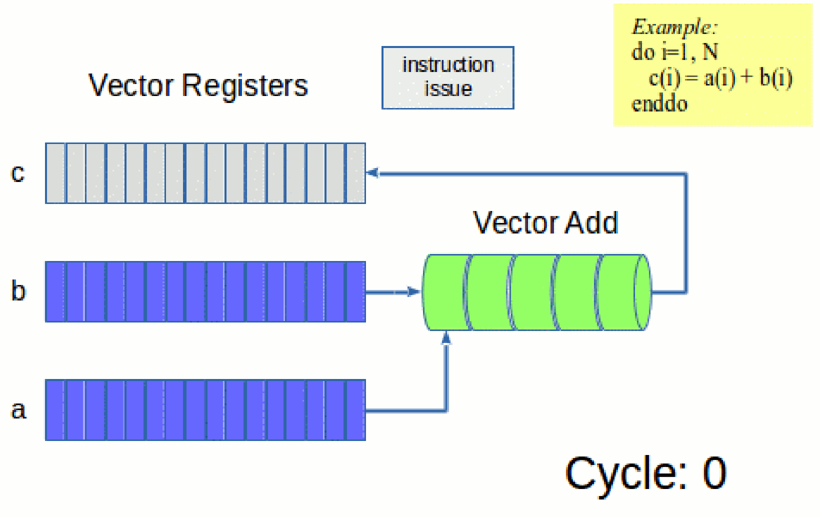
RISC-V ISA <u>VECTOR EXTENSION</u>

RISC-V decided for a VECTOR ISA

- Not a SIMD ISA!
- Variable vector length
  - Vector length register like SX Aurora
- Vector register size not prescribed
  - VLEN = VREG size in bits, **is not fixed!**
  - Left to the implementation
  - Must be power of 2
  - ELEN ≥ 8 max element size in bits
  - ELEN ≤ VLEN ≤ 65536

Code can be VLEN agnostic! Runs on any impl.

or … CRAY-1 style single pipeline



https://github.com/riscv/riscv-v-spec

RISC-V ISA VECTOR EXTENSION

Adds 32 vector registers v0 – v31 of VLEN bits

- "architectural registers"

Adds 7 unprivileged CSRs
(control / status registers)

Remember: vtype encodes

- SEW : single element width (8, 16, 32, 64 bits)
- LMUL : vector register grouping (1/8, ¼, ½, 1, 2, 4, 8)

Masked instructions use v0 as mask

- Mask bits are laid out contiguously in register

Supports precise exceptions

Name	Description
vstart	Vector start position
vxsat	Fixed-Point Saturate Flag
vxrm	Fixed-Point Rounding Mode
vcsr	Vector control and status register
vl	Vector length
vtype	Vector data type register
vlenb	VLEN/8 (vector register length in bytes)

RISC-V ISA VECTOR EXTENSION

- Adds 32 vector registers v0 – v31 of VLEN bits
  - "architectural registers"

- Adds 7 unprivileged CSRs (control / status registers)

  Remember: vtype encodes
  - SEW : single element width (8, 16, 32, 64 bits)
  - LMUL : vector register grouping (1/8, ¼, ½, 1, 2, 4, 8)

- Masked instructions use v0 as mask
  - Mask bits are laid out contiguously in register

- Supports precise exceptions

SX-Aurora

64 registers

Only 64, 32 bits
No register grouping

16 vector mask re
gisters

Imprecise exceptions

SEW AND LMUL

```
VLEN=128b      LMUL=1

Byte         F E D C B A 9 8 7 6 5 4 3 2 1 0

SEW=8b       F E D C B A 9 8 7 6 5 4 3 2 1 0
SEW=16b        7   6   5   4   3   2   1   0
SEW=32b            3       2       1       0
SEW=64b                    1               0
```

```
Example, VLEN=128b, LMUL=1/4

Byte         F E D C B A 9 8 7 6 5 4 3 2 1 0

SEW=8b       - - - - - - - - - - - - 3 2 1 0
SEW=16b        -   -   -   -   -   -   1   0
SEW=32b            -       -       -       0
```

```
VLEN=128b, SEW=32b, LMUL=2

Byte         F E D C B A 9 8 7 6 5 4 3 2 1 0
v2*n               3       2       1       0
v2*n+1             7       6       5       4
```

RISC-V V ISA: VSETVLI

Setting SEW, LMUL, VL, ...

vsetvli rd, rs, eN, mM, tP, mP

- rd : destination register with the actually selected vector length
- rs : source register with **desired vector length** (AVL)
- eN : element width (SEW), eg. e16, e64, e8
- mM : vector group multiplier (LMUL): m1, m2, m4, mf4
- tP : tail handling, ta: tail agnostic, tu: tail undisturbed
- mP : masked elements handling: ma: agnostic, mu: undisturbed

Strip-mining

Works for any VLEN implementation

RISC-V V ISA: VSETVLI

Setting SEW, LMUL, VL, ...

```
vsetvli rd, rs, eN, mM, tP, mP
```

- rd : destination register with the actually selected vector length
- rs : source register with **desired vector length** (AVL)
- eN : element width (SEW), eg. e16, e64, e8
- mM : vector group multiplier (LMUL): m1, m2, m4, mf4
- tP : tail handling, ta: tail agnostic, tu: tail undisturbed
- mP : masked elements handling: ma: agnostic, mu: undisturbed

Strip-mining

Works for any VLEN implementation

SX-Aurora

```
lvl %s42
```

load vector length

RISC-V V ISA: VECTOR OPERATIONS

- Strip-mined loops: no remainder handling
- Strided load / store
- Gather / Scatter
- Vector ops corresponding to each scalar op (arithmetic, logic, shifts, …)
- Reduction instructions (sum, min, max, and, or, …)
- Merge (content of two VREGs according to mask)
- Compress *vcompress* / Expand *vext* (masked)
- Shuffle *vrgather*
- Slide *vslideup*, *vslidedown*, …

RISC-V V ISA: VECTOR OPERATIONS

- Strip-mined loops: no remainder handling
- Strided load / store
- Gather / Scatter
- Vector ops corresponding to each scalar op (arithmetic, logic, shifts, ...)
- Reduction instructions (sum, min, max, and, or, ...)
- Merge (content of two VREGs according to mask)
- Compress *vcompress* / Expand *vext* (masked)
- Shuffle *vrgather* *--> only SX-Aurora VE30*
- Slide *vslideup*, *vslidedown*, ... *--> only SX-Aurora VE30*

RISC-V V ISA: VECTOR OPERATIONS

Vector Gather:

```
vsetvli zero, x[vl], e32, m1, ta, ma
vloxei64.v vd, (x[base]), v[bindex]
```

```
for (size_t i = 0; i < vl; i++) {
  res[i] = *(float32_t*)(bindex[i] + (char*)base);
}
```

- Loads data indirectly, using base address and index stored in vector register
- Size of index elements can be 8, 16, 32, 64 bits!
- Can gather the entire vector in one operation!

SX-Aurora uses addresses instead of indices, *vgt* can gather max 256 elements!

A SIMPLE EXAMPLE

```c
void vadd (double* A, double* B,
   double* C, int size) {
  for (int i = 0; i < size; i++)
      C[i] = A[i] + B[i];
}
```

```
loop:
    vsetvli a7, a4, e64, m1, ta, ma
    ...
    vle64.v v8, (t2)
    vle64.v v10, (t1)
    vfadd.vv v8, v8, v10
    vse64.v v8, (t0)
    ...
    bnez a7, loop
```

RVV

```
loop:
    ...
    lvl     %s60
    vld     %v63,8,%s61     # *(A)
    ...
    vld     %v62,8,%s55     # *(B)
    vfadd.d     %v61,%v63,%v62
    vst     %v61,8,%s61     # *(C)
    brge.w    0,%s56,done
    mins.w.sx    %s60,%s56,%s60
    br.l    loop
```

SX-Aurora

SUMMARY

- RISC-V ISA is simple, extensible
- Vector extension looks complete and very flexible
- Supports many data types
- Can implement short vectors as well as long vectors
- Same machine code runs on any (comparably complete) vector implement ation
  - Long vector code will run well on short vectors.
  - Investment in porting to long vectors is protected!
- Pros and Cons compared to SX-Aurora
  but advantages outweigh disadvantages.

openchip
Enabling the Digital World

THANK YOU