Lecture 16 – RISC-V Vectors

Krste Asanovic
Electrical Engineering and Computer Sciences
University of California at Berkeley

http://www.eecs.berkeley.edu/~krste
http://inst.eecs.berkeley.edu/~cs152
Last Time in Lecture 16

GPU architecture

- Evolved from graphics-only, to more general-purpose computing
- GPUs programmed as attached accelerators, with software required to separate GPU from CPU code, move memory
- Many cores, each with many lanes
  - thousands of lanes on current high-end GPUs
- SIMT model has hardware management of conditional execution
  - code written as scalar code with branches, executed as vector code with predication
New RISC-V “V” Vector Extension

- Being added as a standard extension to the RISC-V ISA
  - An updated form of Cray-style vectors for modern microprocessors

- Today, a short tutorial on current draft standard, v0.7
  - v0.7 is intended to be close to final version of RISC-V vector extension
  - Still a work in progress, so details might change before standardization
  - https://github.com/riscv/riscv-v-spec

- WARNING: Lab 4 uses older version of vector ISA, since new tools not available yet
  - Most concepts carry over, if not programming details
RISC-V Scalar State

Program counter (pc)

32x32/64-bit integer registers (x0-x31)
• x0 always contains a 0

Floating-point (FP), adds 32 registers (f0-f31)
• each can contain a single- or double-precision FP value (32-bit or 64-bit IEEE FP)

FP status register (fcsr), used for FP rounding mode & exception reporting

ISA string options:
• RV32I (XLEN=32, no FP)
• RV32IF (XLEN=32, FLEN=32)
• RV32ID (XLEN=32, FLEN=64)
• RV64I (XLEN=64, no FP)
• RV64IF (XLEN=64, FLEN=32)
• RV64ID (XLEN=64, FLEN=64)
Vector Extension Additional State

- 32 vector data registers, $v0$–$v31$, each VLEN bits long
- Vector length register $vl$
- Vector type register $vtype$
- Other control registers:
  - $vstart$
    - For trap handling
  - $vrm/vxsat$
    - Fixed-point rounding mode/saturation
    - Also appear in $fcsr$

Vector data registers

$VLEN$ bits per vector register, (implementation-dependent)
**Vector Type Register**

<table>
<thead>
<tr>
<th>Bits</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:0</td>
<td>vlmul[1:0]</td>
</tr>
<tr>
<td>4:2</td>
<td>vsew[2:0]</td>
</tr>
<tr>
<td>6:5</td>
<td>vediv[1:0]</td>
</tr>
<tr>
<td>XLEN-1:7</td>
<td>Reserved (write 0)</td>
</tr>
</tbody>
</table>

- **vsew[2:0]** field encodes standard element width (SEW) in bits of elements in vector register \( (SEW = 8 \times 2^{vsew}) \)
- **vlmul[1:0]** encodes vector register length multiplier \( (LMUL = 2^{vlmul} = 1-8) \)
- **vediv[1:0]** encodes how vector elements are divided into equal sub-elements \( (EDIV = 2^{vediv} = 1-8) \)
Example Vector Register Data Layouts (LMUL=1)

VLEN=32b

VLEN=64b

VLEN=128b

VLEN = 256b
Setting vector configuration, \texttt{vsetvli/vsetvl}

The \texttt{vsetvli} configuration instructions set the \texttt{vtype} register, and also set the \texttt{vl} register, returning the \texttt{vl} value in a scalar register.

\begin{verbatim}
vsetvli rd, rs1, e8 # Set SEW=8, \texttt{vl}=\min(\texttt{VLEN/SEW},rs1), \texttt{rd}=\texttt{vl}
\end{verbatim}

\texttt{vtype} parameters (SEW,LMUL,EDIV) encoded as immediate in instruction.

Requested application vector length

Resulting machine vector length setting

\textbf{Instruction encoding}

Usually use immediate form, \texttt{vsetvli}, to set \texttt{vtype} parameters.

The register version \texttt{vsetvl} is usually used for context save/restore.
The first scalar register argument, rs1, is the requested application vector length (AVL)

The type argument (either immediate or second register) indicates how the vector registers should be configured
- Configuration includes size of each element

The vector length is set to the minimum of requested AVL and the maximum supported vector length (VLMAX) in the new configuration
- VLMAX = LMUL*VLEN/SEW
- vl = min(AVL, VLMAX)

The value placed in vl is also written to the scalar destination register rd
Simple stripmined vector `memcpy` example

```c
# void *memcpy(void* dest, const void* src, size_t n)
# a0=dest, a1=src, a2=n
#
memcpy:
    mv a3, a0  # Copy destination
loop:
    vsetvli t0, a2, e8  # Vectors of 8b
    vlb.v v0, (a1)
    add a1, a1, t0
    sub a2, a2, t0
    vsb.v v0, (a3)
    add a3, a3, t0
    bnez a2, loop
    ret
```

Set configuration, calculate vector strip length

Unit-stride vector load bytes

Unit-stride vector store bytes

Binary machine code can run on machines with any VLEN!
Vector Load Instructions

- **unit-stride**
- **strided**
- **indexed**

- **Scalar stride (bytes)**
- **Vector of offsets (bytes)**
- **Scalar base address**
- **Vector destination**
Vector Store Instructions

Vector store data
Vector Unit-Stride Loads/Stores

# vd destination, rs1 base address, vm is mask encoding (v0.t or <missing>)
vlb.v    vd, (rs1), vm    # 8b signed
vlh.v    vd, (rs1), vm    # 16b signed
vlw.v    vd, (rs1), vm    # 32b signed
vlbu.v    vd, (rs1), vm    # 8b unsigned
vlhu.v    vd, (rs1), vm    # 16b unsigned
vlwu.v    vd, (rs1), vm    # 32b unsigned
vle.v    vd, (rs1), vm    # SEW

# vs3 store data, rs1 base address, vm is mask encoding (v0.t or <missing>)
vsb.v    vs3, (rs1), vm    # 8b store
vsh.v    vs3, (rs1), vm    # 16b store
vsw.v    vs3, (rs1), vm    # 32b store
vse.v    vs3, (rs1), vm    # SEW store
Vector Strided Load/Store Instructions

```
# vd destination, rs1 base address, rs2 byte stride
vlsb.v  vd, (rs1), rs2, vm  # 8b
vlsh.v  vd, (rs1), rs2, vm  # 16b
vlsw.v  vd, (rs1), rs2, vm  # 32b

vlbsu.v vd, (rs1), rs2, vm  # unsigned 8b
vlbsh.v vd, (rs1), rs2, vm  # unsigned 16b
vlbsw.v vd, (rs1), rs2, vm  # unsigned 32b

vlse.v  vd, (rs1), rs2, vm  # SEW

# vs3 store data, rs1 base address, rs2 byte stride
vssb.v  vs3, (rs1), rs2, vm  # 8b
vssh.v  vs3, (rs1), rs2, vm  # 16b
vssw.v  vs3, (rs1), rs2, vm  # 32b
vsse.v  vs3, (rs1), rs2, vm  # SEW
```
Vector Indexed Loads/Stores

# vd destination, rs1 base address, vs2 indices
vlxb.v  vd, (rs1), vs2, vm # 8b
vlxh.v  vd, (rs1), vs2, vm # 16b
vlxw.v  vd, (rs1), vs2, vm # 32b
vlxbu.v vd, (rs1), vs2, vm # 8b unsigned
vlxhu.v vd, (rs1), vs2, vm # 16b unsigned
vlxwu.v vd, (rs1), vs2, vm # 32b unsigned
vlxe.v  vd, (rs1), vs2, vm # SEW

# Vector ordered-indexed store instructions
# vs3 store data, rs1 base address, vs2 indices
vsxb.v  vs3, (rs1), vs2, vm # 8b
vsxh.v  vs3, (rs1), vs2, vm # 16b
vsxw.v  vs3, (rs1), vs2, vm # 32b
vsxe.v  vs3, (rs1), vs2, vm # SEW

# Vector unordered-indexed store instructions
vsuxb.v vs3, (rs1), vs2, vm # 8b
vsuxh.v vs3, (rs1), vs2, vm # 16b
vsuxw.v vs3, (rs1), vs2, vm # 32b
vsuxe.v vs3, (rs1), vs2, vm # SEW
Vector Length Multiplier, LMUL

- Gives fewer but longer vector registers
- Set by `vlmul[1:0]` field in `vtype` during `setvli`

LMUL=2

<table>
<thead>
<tr>
<th>F</th>
<th>E</th>
<th>D</th>
<th>C</th>
<th>B</th>
<th>A</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>
<th>Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td><code>v2 * n + 0</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>7</td>
<td>6</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td><code>v2 * n + 1</code></td>
</tr>
</tbody>
</table>

LMUL=4

<table>
<thead>
<tr>
<th>F</th>
<th>E</th>
<th>D</th>
<th>C</th>
<th>B</th>
<th>A</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>
<th>Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>9</td>
<td>8</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>B</td>
<td>A</td>
<td>3</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>D</td>
<td>C</td>
<td>5</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>F</td>
<td>E</td>
<td>7</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6</td>
</tr>
</tbody>
</table>
LMUL=8 stripmined vector memcpy example

Binary machine code can run on machines with any VLEN!
Mixed-Width Loops

- Have different element widths in one loop, even in one instruction
- Want same number of elements in each vector register, even if different bits/element
- Solution: Keep SEW/LMUL constant
VLEN=256b, SLEN=128b

**SEW=8b, LMUL=1, VMAX=32**

| 1F | 1E | 1D | 1C | 1B | 1A | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | F | E | D | C | B | A | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 1F | 1E | 1D | 1C | 1B | 1A | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | F | E | D | C | B | A | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

\[ \text{v1} \times n + 0 \]

**SEW=16b, LMUL=2, VMAX=32**

| 1F | 1E | 1D | 1C | 1B | 1A | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | F | E | D | C | B | A | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 1F | 1E | 1D | 1C | 1B | 1A | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | F | E | D | C | B | A | 9 | 8 |

\[ \text{v2} \times n + 0 \]

\[ \text{v2} \times n + 1 \]

**SEW=32b, LMUL=4, VMAX=32**

| 1F | 1E | 1D | 1C | 1B | 1A | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | F | E | D | C | B | A | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 1F | 1E | 1D | 1C | 1B | 1A | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | F | E | D | C | B | A | 9 |

\[ \text{v4} \times n + 0 \]

\[ \text{v4} \times n + 1 \]

\[ \text{v4} \times n + 2 \]

\[ \text{v4} \times n + 3 \]

**SEW=64b, LMUL=8, VMAX=32**

| 1F | 1E | 1D | 1C | 1B | 1A | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | F | E | D | C | B | A | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 1F | 1E | 1D | 1C | 1B | 1A | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | F | E | D | C | B | A | 9 |

\[ \text{v8} \times n + 0 \]

\[ \text{v8} \times n + 1 \]

\[ \text{v8} \times n + 2 \]

\[ \text{v8} \times n + 3 \]

\[ \text{v8} \times n + 4 \]

\[ \text{v8} \times n + 5 \]

\[ \text{v8} \times n + 6 \]

\[ \text{v8} \times n + 7 \]
CS152 Administrivia

- PS 4 due Friday April 5 in Section
- Lab 4 out on Friday
- Lab 3 due Monday April 8
CS252 Administrivia

Next week readings: Cray-1, VLIW & Trace Scheduling