SW Toolchain for RISC-V Vector Extensions

Roger Ferrer Ibáñez
roger.ferrer@bsc.es

May 3rd 2022
Acknowledgements
### RISC-V Vector Extension (RVV)

#### Physical view

<table>
<thead>
<tr>
<th></th>
<th>V4</th>
<th>V5</th>
<th>V6</th>
<th>V7</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEW</td>
<td>64b</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VLEN</td>
<td>128b</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Logical view

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>vl</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VMAX</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Register group (LMUL=4)

- VLEN=128b
- SEW=64b
- vl=7
- VMAX=8

#### Physical view

<table>
<thead>
<tr>
<th></th>
<th>V8</th>
<th>V9</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEW</td>
<td>32b</td>
<td></td>
</tr>
</tbody>
</table>

#### Logical view

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>vl</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VMAX</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Register group (LMUL=2)

- VLEN=128b
- SEW=32b
- vl=7
- VMAX=8

#### vtype = <SEW, LMUL, policy>
Flexibility

Photo by Benn McGuinness on Unsplash
Challenges in code generation
RVV Architectural State and Instructions

\[
\begin{align*}
%dc & = \text{fadd } <2 \times \text{double}> %da, %db \\
%sc & = \text{fadd } <4 \times \text{float}> %sa, %sb \\
%sc2 & = \text{fadd } <8 \times \text{float}> %sa2, %sb2 \\
%sch & = \text{fadd } <2 \times \text{float}> %sah, %sbh
\end{align*}
\]

\text{VLEN=128b} \\
\rightarrow \text{vl=2, sew=64, lmul=1} \\
\rightarrow \text{vl=4, sew=32, lmul=1} \\
\rightarrow \text{vl=8, sew=32, lmul=2} \\
\rightarrow \text{vl=2, sew=32, lmul=1/2}

\text{vfadd.vv}
Current approach

\[
\%sc \quad = \quad \text{fadd} \quad <4 \times \text{float}> \quad \%sa, \quad \%sb \quad \quad \text{VLEN}=128b
\]

\[
\%3:vr \quad = \quad \text{nopexcept} \quad \text{PseudoVFADD}\_VV\_M1 \quad \%0:vr, \quad \%1:vr, \quad -1, \quad 5, \quad \text{implicit} \quad \$\text{frm}
\]

\[
\text{vl}=\text{VLMAX} \quad \quad \text{vtype}=\langle\text{sew}=32,\text{lmul}=1\rangle
\]
What about setting the context

%3:vr = nophexcept PseudoVFADD_VV_M1 %0:vr, %1:vr, -1, 5, implicit $frm

dead %4:gpr = PseudoVSETVLIX0 $x0, 80, implicit-def $vl, implicit-def $vtype

%3:vr = nophexcept PseudoVFADD_VV_M1 %0:vr, %1:vr, -1, 5, implicit $frm, implicit $vl, implicit $vtype
Challenges that impact the user of RVV
Intrinsics
Vector Predication

Scalar operation (add two double precision values)

\[
\%sc = \text{fadd} \text{ double } \%sa, \%sb
\]

Element-wise extension to whole vectors (add two double precision vector values)

\[
\%vc = \text{fadd} \text{ <8 x double> } \%va, \%vb \\
\%vla.c = \text{fadd} \text{ <vscale x 1 x double> } \%vla.a, \%vla.b
\]

Vector Predication allows us to specify mask and vector length operands

\[
\%vc = \text{call} \text{ <8 x double> } @\text{llvm.vp.fadd.nxv1f64}( \\
\text{<8 x double>} \%vla.a, \\
\text{<8 x double>} \%vla.b, \\
\text{<8 x i1>} \%mask, \text{i32 }\%vl)
\]

\[
\%vla.c = \text{call} \text{ <vscale x 1 x double> } @\text{llvm.vp.fadd.nxv1f64}( \\
\text{<vscale x 1 x double>} \%vla.a, \\
\text{<vscale x 1 x double>} \%vla.b, \\
\text{<vscale x 1 x i1>} \%mask, \text{i32 }\%vl)
\]
Loop Vectorisation at EPI

Typical scheme (simplified)

- overlap check
- trip-count check
- **full** vector loop + mask
- scalar/epilog loop

Tail folding

- overlap check
- **full** vector loop + mask
- scalar/epilog loop

Tail folding + vector length

- overlap check
- **vl** vector loop + mask
- scalar/epilog loop

Hybrid mode

- overlap check
- trip-count check
- **full** vector loop + mask
- vector epilog (vl but no loop)
- scalar/epilog loop

Vector Predication
Example DAXPY kernel

```c
void daxpy(double a,
            double * restrict dx,
            double * restrict dy,
            int n) {
    for (int i = 0; i < n; i++) {
        dy[i] += a * dx[i];
    }
}
```

You can try it at
https://repo.hca.bsc.es/epic/z/iBdt4p
Thank you!

The European Processor Initiative (EPI) has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement EPI-SGA1: 826647 and under EPI-SGA2: 101036168. Please see http://www.european-processor-initiative.eu for more information.

The European PILOT project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No.101034126. The JU receives support from the European Union’s Horizon 2020 research and innovation programme and Spain, Italy, Switzerland, Germany, France, Greece, Sweden, Croatia and Turkey.

The MEEP project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 946002. The JU receives support from the European Union’s Horizon 2020 research and innovation programme and Spain, Croatia, Turkey

roger.ferrer@bsc.es