Simulation Evaluation of Chaining Implementations for the RISC-V Vector Extension

Zhen Wei <zhenwei.tw@gmail.com>, National Taiwan University
Wei-Chung Hsu <hsuwc@csie.ntu.edu.tw>, National Taiwan University
RISC-V “V” Extension

- An instruction set explicitly supports “Data-Level Parallelism” in the program
  - A single vector instruction can launch several data operations

- A more scalable SIMD (Single Instruction Multiple Data) architecture
  - No ISA changes needed when the micro-architecture can afford more SIMD lanes
  - From microprocessors to supercomputers, RISC-V “V” extension [1] can scale well
RISC-V “V" Extension

- RISC-V “V" extension (RVV) follows vector-length agnostic (VLA) architecture
  - Vectorized code could run in any implementation with different vector lengths
  - VLA enables CPU architects to build implementations with longer vector lengths
- VLA decouples ISA from implementation
  - Vector length can be long, yet the number of execution lanes could be small
  - A vector instruction may take multiple or even many cycles to complete

![Diagram showing function unit latency and instruction issue]

- VMUL v1, v2, v3
- VADD v4, v1, v5
- VSHT v6, v4, v7

Function unit latency: "VL / (# of execution lanes)" cycles

Time
Chaining [2] is used to allow dependent vector instructions to start execution early

- Partial results of vector instruction could be forwarded to the next dependent instruction
- In the following example, we assume that $X_i = 3$ and $Y_i = 4$ for all $i$, the implementation w/ chaining takes 15 cycles, w/o chaining takes 21 cycles, that is 40% of speedup.

<table>
<thead>
<tr>
<th>Code sequence</th>
<th>Implementation w/o chaining</th>
<th>Implementation w/ chaining</th>
</tr>
</thead>
<tbody>
<tr>
<td>VMUL v1, v2, v3</td>
<td>$X_1 \rightarrow Y_1 \rightarrow X_2 \rightarrow Y_2 \rightarrow X_3 \rightarrow Y_3$</td>
<td>$X_1 \rightarrow Y_1 \rightarrow X_2 \rightarrow Y_2$</td>
</tr>
<tr>
<td>VADD v4, v1, v5</td>
<td>$VADD$ can be issued at cycle $X_1 + 1$</td>
<td>$X_3 \rightarrow Y_3$</td>
</tr>
<tr>
<td>VSHT v6, v4, v7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Vector Micro-architecture

- Vector OPeration unit (VOP)
  - VOP produces many “operations” according to the vector instruction and current vector length
  - E.g. an implementation with vector register size 512 bits and 2 * 64 bits execution lanes is going to require $\frac{512}{(2 \times 64)} = 4$ operations

- Vector Function Unit (VFU)
  - VFUs are responsible for executing operations dispatched by VOP
  - “Number of execution lanes” is a parameterized feature in our simulation

\[ \text{VALU: Vector integer ALU handles normal arithmetic, logical and shift operations} \]
\[ \text{VMUL: Vector integer MUL handles multiply operations} \]
Vector Micro-architecture

- Vector function unit (VFU)
  - Before chaining is implemented, forwarding data paths must be in place inside the vector unit.
  - We assume full forwarding paths in our simulation, where the result of any vector function unit can be forwarded to other vector function units.

Forwarding paths inside the vector unit

- VALU: Vector integer ALU handles normal arithmetic, logical and shift operations.
- VMUL: Vector integer MUL handles multiply operations.
Vector Micro-architecture

● Example code
  ○ VADD  v1, v2, v3
  ○ VMUL  v4, v1, v5
  ○ VSHT  v6, v5, v7

● Configurations
  ○ Vector register size: **128** bits
  ○ Vector execution capability: **64** bits
  ○ VOP will produce **2** operations for **1** vector instruction

※VALU: Vector integer ALU handles normal arithmetic, logical and shift operations
※VMUL: Vector integer MUL handles multiply operations
Vector Micro-architecture

- Example code
  - VADD \( v1, v2, v3 \)
  - VMUL \( v4, v1, v5 \)
  - VSHT \( v6, v5, v7 \)

- Configurations
  - Vector register size: 128 bits
  - Vector execution capability: 64 bits
  - VOP will produce 2 operations for 1 vector instruction

Valu: Vector integer ALU handles normal arithmetic, logical and shift operations

VMUL: Vector integer MUL handles multiply operations

VOP: Vector operation
Vector Micro-architecture

- Example code
  - VADD \( v1, v2, v3 \)
  - VMUL \( v4, v1, v5 \)
  - VSHT \( v6, v5, v7 \)

- Configurations
  - Vector register size: \textbf{128} bits
  - Vector execution capability: \textbf{64} bits
  - VOP will produce 2 operations for 1 vector instruction

※VALU: Vector integer ALU handles normal arithmetic, logical and shift operations
※VMUL: Vector integer MUL handles multiply operations
Vector Micro-architecture

- Example code
  - \texttt{VADD} v1, v2, v3
  - \texttt{VMUL} v4, v1, v5
  - \texttt{VSHT} v6, v5, v7

- Configurations
  - Vector register size: \textbf{128} bits
  - Vector execution capability: \textbf{64} bits
  - VOP will produce \textbf{2} operations for \textbf{1} vector instruction

\begin{itemize}
  \item \texttt{VALU}: Vector integer ALU handles normal arithmetic, logical and shift operations
  \item \texttt{VMUL}: Vector integer MUL handles multiply operations
\end{itemize}
Vector Micro-architecture

- Example code
  - VADD \( v_1, v_2, v_3 \)
  - VMUL \( v_4, v_1, v_5 \)
  - VSHT \( v_6, v_5, v_7 \)

- Configurations
  - Vector register size: 128 bits
  - Vector execution capability: 64 bits
  - VOP will produce 2 operations for 1 vector instruction

![Diagram of vector micro-architecture]

※VALU: Vector integer ALU handles normal arithmetic, logical and shift operations
※VMUL: Vector integer MUL handles multiply operations

March 12, 2019 @ RISC-V Workshop Taiwan
Vector Micro-architecture

- Chaining implementations
  - Vector function unit chaining
  - Memory chaining
    - In addition to vector function unit chaining, CRAY vector supercomputers, such as Cray-1/Cray-XMP/Cray-YMP, also support memory chaining

- Two chaining scenarios are separately considered
  - Restricted chaining: supports only vector function unit chaining, no memory chaining
  - Full chaining: supports both vector function unit and memory chaining
Vector Micro-architecture

- Memory chaining
  - Easy for Cray since they don’t use data caches and virtual memory.
  - Unfortunately, data caches and virtual memory are commonplace in modern microprocessors.
  - Challenge to implement memory chaining in cache-based processors:
    - Compared to vector function unit chaining, the completion time of vector load instruction is non-deterministic.
    - Cache misses and TLB misses, which may happen within a vector instruction, are difficult to handle.
    - Additional hardware resources are needed to track many outstanding memory accesses induced by vector load instruction.
Vector Micro-architecture

- Memory unit
  - Scalar and vector memory instructions use the same memory unit
  - When a vector load instruction is ready to issue, VOP will take over the memory unit and dispatch the memory requests (i.e. vector load operations)
  - We also look at the impact of different Miss Status Holding Register (MSHR) resources (# of outstanding memory accesses, # of bus requests) in our simulation

※ Bandwidth of accessing cache needs to be consistent with the capability of VFUs
※ Unblocking cache is implemented and the MSHR (Miss status handling register) resources will be considered in our simulation
Vector Micro-architecture

- Memory chaining implementation
  - When a vector load operation complete, it will lift the corresponding ready bit of the target vector register

- Example
  - When a vector load instruction tries to write v1 register ...

<table>
<thead>
<tr>
<th>Physical Vector Register</th>
<th>Read / Write “Valid”</th>
<th>Read / Write “LChaining”</th>
<th>DataReadyBits</th>
</tr>
</thead>
<tbody>
<tr>
<td>pv0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pv1</td>
<td>True</td>
<td>True</td>
<td>1000 → 1100</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pv31</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Physical vector register state

1. Cache hit!

VLW. OP3 → Address Generation → VLW. OP2 → Access Cache → VLW. OP1 → Tag Compare → Write Back

March 12, 2019 @ RISC-V Workshop Taiwan
### Vector Micro-architecture

- Memory chaining implementation
  - A vector instruction depending on vector load can be issued early to the vector scoreboard
  - ② VOP checks where does the data of source operand come from
    - If ("Memory Chaining"): continue ③
    - else: produce the operation
  - ③ VOP checks if the data of a specific vector operation is ready

---

<table>
<thead>
<tr>
<th>VFU</th>
<th>Valid</th>
<th>Read Operand1</th>
<th>Read Operand2</th>
<th>Operation Produced</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALU1</td>
<td>True</td>
<td>pv1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>VMEM</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Physical Vector Register</th>
<th>Read/Write “Valid”</th>
<th>Read/Write “Chaining”</th>
<th>DataReadyBytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>pv0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pv1</td>
<td>True</td>
<td>True</td>
<td>1100</td>
</tr>
<tr>
<td>pv31</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

Physical vector register state

March 12, 2019 @ RISC-V Workshop Taiwan
Vector Micro-architecture

- Overall micro-architecture

### Configurations

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector register size</td>
<td>512 ~ 4096 bits</td>
</tr>
<tr>
<td>Capability of vector unit</td>
<td>64 ~ 512 bits</td>
</tr>
<tr>
<td>Cache miss penalty</td>
<td>12 ~ 96 cycles</td>
</tr>
<tr>
<td># of outstanding memory access</td>
<td>8, 256</td>
</tr>
<tr>
<td># of bus requests</td>
<td>2, 32</td>
</tr>
</tbody>
</table>
Performance Evaluation

- Benchmark - Matrix multiplication [3]
  - Matrix A (64*64) and Matrix B (64*256) are used in our evaluation

```java
for (int m = 0; m < M; m++) {
    for (int k = 0; k < K; k += VL) {
        sum = 0
        for (int n = 0; n < N; n++) {
            A_{m,n} = vectorLoad.s(&A[m][n]);
            B_{n, [k:k+VL]} = vectorLoad.v(&B[n][k]);
            sum += A_{m,n} * B_{n, [k:k+VL]};
        }
        vectorStore.v(sum, &C[m][k]);
    }
}
```
Performance Evaluation

- Restricted chaining
  - Vector function unit chaining gives us at least 15% performance improvement
  - If the vector registers are grouped, which is offered by RVV, vector function unit chaining can give us up to 59% performance improvement

Based on the Unoptimized Code
Performance Evaluation

- Full chaining
  - Full chaining gives us at least 30% performance improvement
  - If the vector registers are grouped, which is offered by RVV, full chaining can give us up to 69% performance improvement

Based on the Unoptimized Code
Performance Evaluation

- Impact of different Vector Execution Capabilities (VEC)
  - When VEC is weaker, the impact of chaining is greater
  - The vector register size and VEC need to be considered together

\[ \text{Execution time} = (x_1) + (x_2 + y_2) + 1 \]

- VMUL v1, v2, v3
- VADD v4, v1, v5

Based on the Unoptimized Code
Performance Evaluation

- Impact of different cache Miss Penalties (MP)
  - Chaining is more crucial to performance when the memory latency is smaller, which is more sensitive to the delay issuing of vector instructions

\[
\text{Execution time} = (x_1 + \text{Memory latency}) + (x_2 + y_2) + 1
\]

\[
\text{VLW v1} \quad \begin{array}{c}
X_1 \\
\text{Memory latency}
\end{array} \quad \begin{array}{c}
Y_1 \\
X_2 \\
y_2
\end{array}
\]

\[
\text{VADD v4, v1, v5}
\]

- Based on the Unoptimized Code
Code Optimizations for No-chaining

- Optimize the code for no chaining mechanism
  - Loop unrolling is used to produce many independent vector instructions for scheduling
  - Software pipelining helps us issue the vector load instruction in the next iteration early
  - Vector instructions inside a chime can be executed at the same time

innermost loop

```c
vmul v3, v1, v2
vlw.s v1, &A[m][n]  
vlw.v v2, &B[n][k]  
n++
vlw.s v5, &A[m][n]  
vlw.v v6, &B[n][k]  
n++

vadd v4, v3, v4
vmul v7, v5, v6
vlw.s v5, &A[m][n]  
vlw.v v6, &B[n][k]  
n++
vadd v4, v7, v4
```
Code Optimizations for Chaining

- Optimize the code for chaining
  - Similar optimization techniques also work for chaining
  - Chaining favors scheduling data dependent instructions together
    - E.g. “vadd” can be chained to the “vmul”

```c
vlw.s v1, &A[m][n]
vlw.v v2, &B[n][k]
n++
vlw.s v5, &A[m][n]
vlw.v v6, &B[n][k]
n++
vlw.s v1, &A[m][n]
vlw.v v2, &B[n][k]
n++
vlw.s v5, &A[m][n]
vlw.v v6, &B[n][k]
n++
vmul v3, v1, v2
vadd v4, v4, v3
vlw.s v1, &A[m][n]
vlw.v v2, &B[n][k]
n++
vmul v7, v5, v6
vadd v4, v7, v4
vlw.s v5, &A[m][n]
vlw.v v6, &B[n][k]
n++
vmul v3, v1, v2
vadd v4, v4, v3
vmul v7, v5, v6
vadd v4, v7, v4
```
Performance Evaluation

- After code optimization
  - Code optimizations can help the no chaining implementation to catch up the performance gap with chaining
    - However, not all loops can be optimized this way
    - When the vector register size increases, more unrolling or loop fusion are called for.
Performance Evaluation

- Impact of MSHR resources
  - Performance may be bounded by the MSHR resources
    - Previous experiments are based on the ideal scenario with sufficient MSHR resources

Performance of Unoptimized Code

- MSHR (2, 8)
- MSHR (32, 256)
- Gap

Performance of Optimized Code

- MSHR (2, 8)
- MSHR (32, 256)
- Gap
Summary

● Full or restricted chaining can help speeding up unoptimized code
  ○ Chaining is more effective when your vector register is longer (if your code has enough DP)

● Code optimizations allow the implementation w/o chaining to catchup with chaining
  ○ However, average vector length (100-200) in real applications may not get full benefit of unrolling/software pipelining, especially when the vector register size is longer
  ○ Full or restricted chaining can naturally speed up general loops and further performance gain can be obtained by additional optimizations

● For a cost-effective vector micro-architecture, your design must also consider the following:
  ○ MSHR resources, cache miss penalty and number of cache port
  ○ Vector register size and the execution capability of vector function unit
  ○ Effective code optimizations (not all loops can be optimized to overcome the lack of chaining)
Reference


Thanks for your attention!