Understanding Ascalon's Vector Unit: Enhancing RISC-V Vector Performance (Under Review)

Overview

Ascalon, Tenstorrent's microarchitecture, includes advanced hardware to optimize RISC-V Vector (RVV) code execution. This article provides a teaching overview of the vector unit's design and functionality, particularly its front-end and datapath improvements, highlighting how these innovations deliver high performance.

Vector Front-End: Decoding Complexity Into Efficiency

The front-end of Ascalon’s vector unit is designed to handle the unique challenges presented by RVV, an extension to the RISC-V ISA that supports variable-length vector processing. Unlike traditional scalar instructions, RVV instructions can process multiple addresses and registers simultaneously, with behavior influenced by the `vtype` CSR (Control and Status Register). Here's how Ascalon efficiently manages this complexity:

Vector Cracking

RVV instructions are broken into micro-operations (uOps) to streamline backend consumption. These uOps adhere to strict rules:

- **Single Destination, Multiple Sources**: Each uOp writes one destination register and reads up to five source registers.
- **Memory Access Limits**: A uOp can access up to eight unique memory addresses, but all accesses must have the same size.
- **Resource Constraints**: Temporary register usage is tightly controlled to support small core designs.
- **Mask Register Handling**: The mask register (`v0`) is copied to a temporary when required.

The front-end decodes instructions based on static bits and `vtype` CSR values, not runtime data. Even if vector operations are partially masked (e.g., processing one element while `LMUL=8`), all corresponding uOps are generated.

Decoding Restrictions:

- Instructions are padded with no-operations (NOPs) to fit fixed decoding patterns.
- Complex instructions are only decoded from specific lanes, with stalling when misaligned.

CSR Management

Efficient handling of vector-related CSRs (`vtype` , `vl` , `vcsr`) is crucial for performance. Key features include:

- `vtype` **Renaming**: `vtype` is renamed and stored in a circular buffer with 64 entries to ensure quick recovery during branch mispredictions or load resynchronizations. This avoids pipeline stalls during decoding.
- `vl` **Handling**: The vector length register (`vl`) is renamed in a separate 64-entry space. Writes to `vl` are managed by dedicated uOps, and reads are added explicitly as operands to vector instructions.
- **Pipeline Serialization**: For correctness, the front-end serializes reads/writes to `vcsr` , particularly for rounding modes (`vxrm`) and overflow flags (`vxsat`).

Vector Datapath: High Throughput and Scalability

The Ascalon vector unit achieves high performance through its dual 256-bit symmetric vector datapaths, delivering 512-bit processing per cycle. These datapaths support various operation types and latencies, tailored for both integer and floating-point workloads.

Pipeline Design

The vector pipeline supports:

- **Fixed-Latency Operations**: Logical operations (AND, OR) complete in one cycle, while additions (integer and floating-point) take two cycles. More complex operations, such as integer multiplication, require three cycles, and floating-point multiply-accumulate takes four.
- **Variable-Latency Operations**: Certain non-pipelined operations, like division and square root, are processed separately via a long-latency path to avoid blocking the main pipeline.

Execution Units

Each lane in the vector datapath features duplicated units to efficiently handle widening operations while maintaining latency for non-widening operations. For example:

- **Addition Units**: Integer and floating-point adders are duplicated to process results immediately after the widening network.

Rounding Modes and Exception Handling

- **Rounding Modes**: Floating-point and fixed-point operations use `frm` and `vxrm` CSRs, respectively. Writes to these registers serialize the pipeline as per RISC-V guidance.

- **Sticky Flags**: Arithmetic exceptions do not interrupt execution but set sticky bits in `fflags` (floating-point) or `vxsat` (fixed-point). These flags are read non-speculatively at the head of the reorder buffer (ROB).

Special Functionality

Reductions

- **Ordered Reductions**: Widening and non-widening reductions are processed serially, one element at a time.
- **Unordered Reductions**: Ascalon supports horizontal reductions (e.g., summing all elements of a vector) using efficient binary reduction trees and intermediate values.

Crossbar Operations

The vector unit's two crossbars enable flexible data movement:

- **Full Connectivity**: Each crossbar supports 2x256-to-256-bit operations, allowing seamless execution of short-vector operations.
- **Segmented Load/Store**: Segment operations use a special crossbar mode, reducing complexity for multi-element accesses.

Cryptography

Ascalon supports all RVV cryptographic extensions except ShangMi, ensuring robust performance for secure workloads.

Performance Characteristics

Throughput and Latency

Performance varies by operation type and vector configuration (`LMUL` , `SEW` , etc.). For example:

- Simple operations like `vadd.vv` achieve low latency with minimal resource usage.
- More complex instructions, such as `vfdiv.vv` , exhibit longer latencies due to serial dependencies.

Long-Latency Operations

Operations exceeding four cycles (e.g., division, square root) are managed via a separate scheduler. Only one such operation per scheduler is active at a time, ensuring smooth pipeline flow.

Conclusion

The Ascalon vector unit represents a significant step forward in RVV processing. Its innovative front-end decoding, efficient datapath design, and support for advanced features like cryptography and reductions make it a powerful solution for high-performance vector workloads. With Ascalon, developers can fully harness the potential of RISC-V's vector extension, delivering exceptional performance across a wide range of applications.