Vector Execution Time

- The execution time of a sequence of vector operations primarily depends on three factors: (1) the length of the operand vectors, (2) structural hazards among the operations, and (3) the data dependences. Given the vector length and the initiation rate, which is the rate at which a vector unit consumes new operands and produces new results, we can compute the time for a single vector instruction. All modern vector computers have vector functional units with multiple parallel pipelines (or lanes) that can produce two or more results per clock cycle, but they may also have some functional units that are not fully pipelined. For simplicity, our VMIPS implementation has one lane with an initiation rate of one element per clock cycle for individual operations. Thus, the execution time in clock cycles for a single vector instruction is-approximately the vector length.

- A convoy, which is the set of vector instructions that could potentially execute together. The instructions in a convoy must not contain any structural hazards; if such hazards were present, the instructions would need to be serialized and initiated in different convoys. To keep the analysis simple, we assume that a convoy of instructions must complete execution before any other instructions (scalar or vector) can begin execution.

- vector instruction sequences with structural hazards, sequences with read-after-write dependency hazards should also be in separate convoys, but chaining allows them to be in the same convoy.

- Chaining allows a vector operation to start as soon as the individual elements of its vector source operand become available: The results from the first functional unit in the chain are "forwarded" to the second functional unit. In practice, we often implement chaining by allowing the processor to read and write a particular vector register at the same time, albeit to different elements.

- Early implementations of chaining worked just like forwarding in scalar pipelining, but this restricted the timing of the source and destination instructions in the chain. Recent implementations use flexible chaining, which allows a vector instruction to chain to essentially any other active vector instruction, assuming that we don't generate a structural hazard.

- To turn convoys into execution time we need a timing metric to estimate the time for a convoy. It is called a chime, which is simply the unit of time taken to execute one convoy. Thus, a vector sequence that consists of m convoys executes in m chimes; for a vector length of n, for VMIPS this is approximately in x n clock cycles.

- Hence, measuring time in chimes is a better approximation for long vectors than for short ones. We will use the chime measurement, rather than clock cycles per result, to indicate explicitly that we are ignoring certain overheads.
- If we know the number of convoys in a vector sequence, we know the execution time in chimes.

Example:

Show how the following code sequence lays out in convoys, assuming a single copy of each vector functional unit:

```
LV              V1,Rx      ;load vector X
MLILVS.D        V2,V1,F0   ;vector-scalar multiply
LV              V3,Ry      ;load vector Y
ADDVV.D         V4,V2,V3   ;add two vectors
SV              V4,Ry      ;store the sum
```

How many chimes will this vector sequence take? How many cycles per FLOP (floating-point operation) are needed, ignoring vector instruction issue overhead?

Answer:

The first convoy starts with the first LV instruction. The MULVS. D is dependent on the first LV, but chaining allows it to be in the same convoy.

The second LV instruction must be in a separate convoy since there is a structural hazard on the load/store unit for the prior LV instruction.

The ADDVV. D is dependent on the second LV, but it can again be in the same convoy via chaining. Finally, the SV has a structural hazard on the LV in the second convoy, so it must go in the third convoy. This analysis leads to the following layout of vector instructions into convoys:

1. LV MULVS.D
2. LV ADDVV.D
3. SV

The sequence requires three convoys. Since the sequence takes three chimes and there are two floating-point operations per result, the number of cycles per FLOP is 1.5 (ignoring any vector instruction issue overhead). Note that, although we allow the LV and MULVS.D both to execute in the first convoy, most vector machines will take two clock cycles to initiate the instructions.

This example shows that the chime approximation is reasonably accurate for long vectors. For example, for 64-element vectors, the time in chimes is 3, so the sequence would take about 64 x 3 or 192 clock cycles. The overhead of issuing convoys in two separate clock cycles would be small.