Vector Execution Time

The execution time of a sequence of vector operations primarily depends on three factors: the length of the operand vectors, structural hazards among the operations, and 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 supercomputers have vector functional units with multiple parallel pipelines (or *lanes*) that can produce two or more results per clock cycle, but 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 for a single vector instruction is approximately the vector length.

To simplify the discussion of vector execution and its timing, we will use the notion of a *convoy,* which is the set of vector instructions that could potentially begin execution together in one clock period. (Although the concept of a convoy is used in vector compilers, no standard terminology exists. Hence, we created the term *convoy.*) The instructions in a convoy *must not* contain any structural or data hazards (though we will relax this later); if such hazards were present, the instructions in the potential convoy would need to be serialized and initiated in different convoys. Placing vector instructions into a convoy is analogous to placing scalar operations into a VLIW instruction. 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. We will relax this in Section G.4 by using a less restrictive, but more complex, method for issuing instructions.

Accompanying the notion of a convoy is a timing metric, called a *chime,* that can be used for estimating the performance of a vector sequence consisting of convoys. A chime is the unit of time taken to execute one convoy. A chime is an approximate measure of execution time for a vector sequence; a chime measurement is independent of vector length. Thus, a vector sequence that consists of m convoys executes in m chimes, and for a vector length of n, this is approximately $m \times n$ clock cycles. A chime approximation ignores some processor-specific overheads, many of which are dependent on vector length. Hence, measuring time in chimes is a better approximation for long vectors. We will use the chime measurement, rather than clock cycles per result, to explicitly indicate that certain overheads are being ignored.

If we know the number of convoys in a vector sequence, we know the execution time in chimes. One source of overhead ignored in measuring chimes is any limitation on initiating multiple vector instructions in a clock cycle. If only one vector instruction can be initiated in a clock cycle (the reality in most vector processors), the chime count will underestimate the actual execution time of a convoy. Because the vector length is typically much greater than the number of instructions in the convoy, we will simply assume that the convoy executes in one chime.

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
MULVS.D   V2,V1,F0    ;vector-scalar multiply
LV        V3,Ry       ;load vector Y
ADDV.D    V4,V2,V3    ;add
SV        Ry,V4       ;store the result
```

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 is occupied by the first `LV` instruction. The `MULVS.D` is dependent on the first `LV`, so it cannot be in the same convoy. The second `LV` instruction can be in the same convoy as the `MULVS.D`. The `ADDV.D` is dependent on the second `LV`, so it must come in yet a third convoy, and finally the `SV` depends on the `ADDV.D`, so it must go in a following convoy. This leads to the following layout of vector instructions into convoys:

1. `LV`
2. `MULVS.D` `LV`
3. `ADDV.D`
4. `SV`

The sequence requires four convoys and hence takes four chimes. Since the sequence takes a total of four chimes and there are two floating-point operations per result, the number of cycles per FLOP is 2 (ignoring any vector instruction issue overhead). Note that although we allow the `MULVS.D` and the `LV` both to execute in convoy 2, most vector machines will take 2 clock cycles to initiate the instructions.

The chime approximation is reasonably accurate for long vectors. For example, for 64-element vectors, the time in chimes is four, so the sequence would take about 256 clock cycles. The overhead of issuing convoy 2 in two separate clocks would be small.

Another source of overhead is far more significant than the issue limitation. The most important source of overhead ignored by the chime model is vector *start-up time*. The start-up time comes from the pipelining latency of the vector operation and is principally determined by how deep the pipeline is for the functional unit used. The start-up time increases the effective time to execute a convoy to more than one chime. Because of our assumption that convoys do not overlap in time, the start-up time delays the execution of subsequent convoys. Of course the instructions in successive convoys have either structural conflicts for some functional unit or are data dependent, so the assumption of no overlap is

reasonable. The actual time to complete a convoy is determined by the sum of the vector length and the start-up time. If vector lengths were infinite, this start-up overhead would be amortized, but finite vector lengths expose it, as the following example shows.

Example Assume the start-up overhead for functional units is shown in Figure G.4.

Show the time that each convoy can begin and the total number of cycles needed. How does the time compare to the chime approximation for a vector of length 64?

Answer Figure G.5 provides the answer in convoys, assuming that the vector length is n. One tricky question is when we assume the vector sequence is done; this determines whether the start-up time of the SV is visible or not. We assume that the instructions following cannot fit in the same convoy, and we have already assumed that convoys do not overlap. Thus the total time is given by the time until the last vector instruction in the last convoy completes. This is an approximation, and the start-up time of the last vector instruction may be seen in some sequences and not in others. For simplicity, we always include it.

The time per result for a vector of length 64 is 4 + (42/64) = 4.65 clock cycles, while the chime approximation would be 4. The execution time with start-up overhead is 1.16 times higher.

Unit	Start-up overhead (cycles)
Load and store unit	12
Multiply unit	7
Add unit	6

Figure G.4 Start-up overhead.

Convoy	Starting time	First-result time	Last-result time
1. LV	0	12	11 + n
2. MULVS.D LV	12 + n	12 + n + 12	23 + 2n
3. ADDV.D	24 + 2n	24 + 2n + 6	29 + 3n
4. SV	30 + 3n	30 + 3n + 12	41 + 4n

Figure G.5 Starting times and first- and last-result times for convoys 1 through 4. The vector length is n.

For simplicity, we will use the chime approximation for running time, incorporating start-up time effects only when we want more detailed performance or to illustrate the benefits of some enhancement. For long vectors, a typical situation, the overhead effect is not that large. Later in the appendix we will explore ways to reduce start-up overhead.

Start-up time for an instruction comes from the pipeline depth for the functional unit implementing that instruction. If the initiation rate is to be kept at 1 clock cycle per result, then

$$\text{Pipeline depth} = \left\lceil \frac{\text{Total functional unit time}}{\text{Clock cycle time}} \right\rceil$$

For example, if an operation takes 10 clock cycles, it must be pipelined 10 deep to achieve an initiation rate of one per clock cycle. Pipeline depth, then, is determined by the complexity of the operation and the clock cycle time of the processor. The pipeline depths of functional units vary widely—from 2 to 20 stages is not uncommon—although the most heavily used units have pipeline depths of 4–8 clock cycles.

For VMIPS, we will use the same pipeline depths as the Cray-1, although latencies in more modern processors have tended to increase, especially for loads. All functional units are fully pipelined. As shown in Figure G.6, pipeline depths are 6 clock cycles for floating-point add and 7 clock cycles for floating-point multiply. On VMIPS, as on most vector processors, independent vector operations using different functional units can issue in the same convoy.

Vector Load-Store Units and Vector Memory Systems

The behavior of the load-store vector unit is significantly more complicated than that of the arithmetic functional units. The start-up time for a load is the time to get the first word from memory into a register. If the rest of the vector can be supplied without stalling, then the vector initiation rate is equal to the rate at which new words are fetched or stored. Unlike simpler functional units, the initiation rate may not necessarily be 1 clock cycle because memory bank stalls can reduce effective throughput.

Operation	Start-up penalty
Vector add	6
Vector multiply	7
Vector divide	20
Vector load	12

Figure G.6 Start-up penalties on VMIPS. These are the start-up penalties in clock cycles for VMIPS vector operations.