Chapter 4

Data-Level Parallelism (DLP) in Vector, SIMD, and GPU Architectures

Part 2: Advanced Vector Architectures
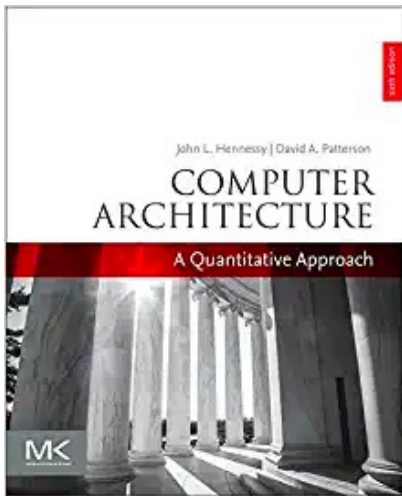
"Who's first?"
"America."
"Who's second?"
"Sir, there is no second."

> -Dialog between two observers of the sailing race later named "The America's Cup" and run every few years -- the inspiration for John Cocke's naming of the IBM research processor as "America." This processor was the precursor to the RS/6000 series and the first superscalar microprocessor.
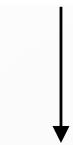
Acknowledgements

- ## Thanks to many sources for slide material

DAXPY (Y = <u>a</u> * <u>X + Y</u>)

Assuming vectors X, Y are length 64

Scalar vs. Vector

```
            LD      F0,a            ;load scalar a
            LV      V1,Rx           ;load vector X
            MULTS   V2,F0,V1        ;vector-scalar mult.
            LV      V3,Ry           ;load vector Y
            ADDV    V4,V2,V3        ;add
            SV      Ry,V4           ;store the result
```

```
            LD      F0,a
            ADDI    R4,Rx,#512      ;last address to load
loop:  LD       F2, 0(Rx)          ;load X(i)
            MULTD   F2,F0,F2        ;a*X(i)
            LD       F4, 0(Ry)      ;load Y(i)
            ADDD    F4,F2,F4        ;a*X(i) + Y(i)
            SD      F4,0(Ry)        ;store into Y(i)
            ADDI    Rx,Rx,#8        ;increment index to X
            ADDI    Ry,Ry,#8        ;increment index to Y
            SUB     R20,R4,Rx       ;compute bound
            BNZ     R20,loop        ;check if done
```

578 (2+9*64) vs.
321 (1+5*64) ops (1.8X)

578 (2+9*64) vs.
6 instructions (96X)

64 operation vectors +
no loop overhead

also 64X fewer pipeline
hazards

Vector Execution Time

- Execution time depends on several factors:
  - Length of operand vectors
  - Structural hazards
  - Data dependencies
  - Pipeline depth → start-up latency (short vs. long vectors?)

- VMIPS functional units consume one element per clock cycle
  - Execution time is approximately the vector length

- Convoy
  - Set of vector instructions that could potentially execute together

Vector Inefficiency

- Must wait for last element of result to be written before starting dependent instruction

Vector Startup

- Vector startup penalty
  - Functional unit latency (time thru pipeline)
  - Dead time or recovery time (time before another vector instruction can start down pipeline)

Vector Execution Time

- Execution time depends on several factors:
  - Length of operand vectors
  - Structural hazards
  - Data dependencies
  - Pipeline depth → start-up latency (short vs. long vectors?)

- VMIPS functional units consume one element per clock cycle
  - Execution time is approximately the vector length

- Convoy
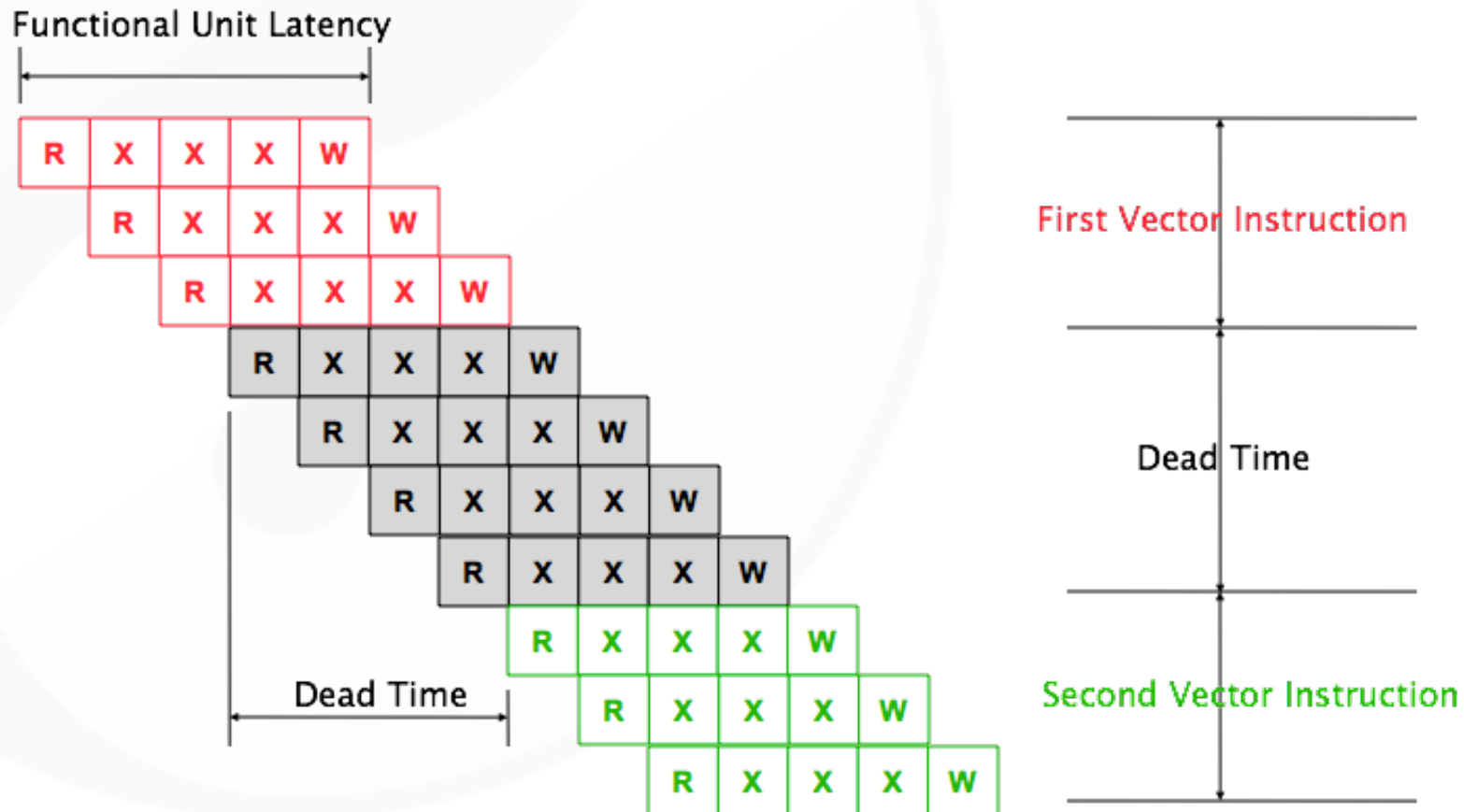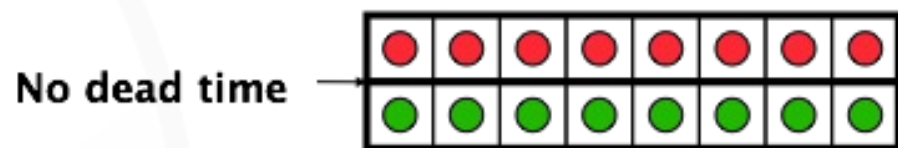 - Set of vector instructions that could potentially execute together

Dead Time and Short Vectors

4 cycles dead time

64 cycles active

Cray C90, Two lanes
4 cycle dead time
Maximum efficiency 94%
with 128 element vectors
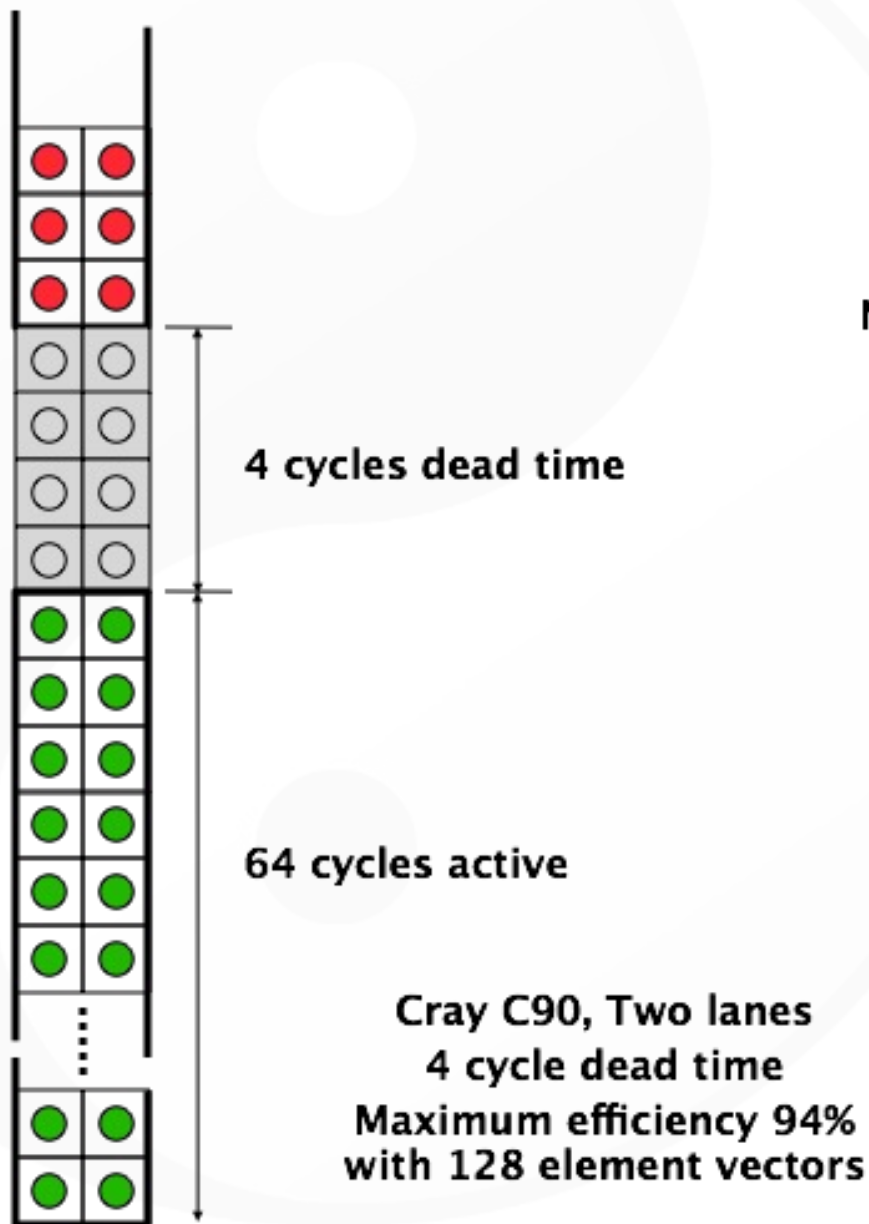
No dead time →

T0, Eight lanes
No dead time
100% efficiency with 8 element vectors

Chimes

- Chime
  - Unit of time to execute one convoy (or a vector operation)
  - m convoys executes in m chimes
  - For vector length of n, requires m x n clock cycles
    - When does this estimation become more accurate? Less accurate?
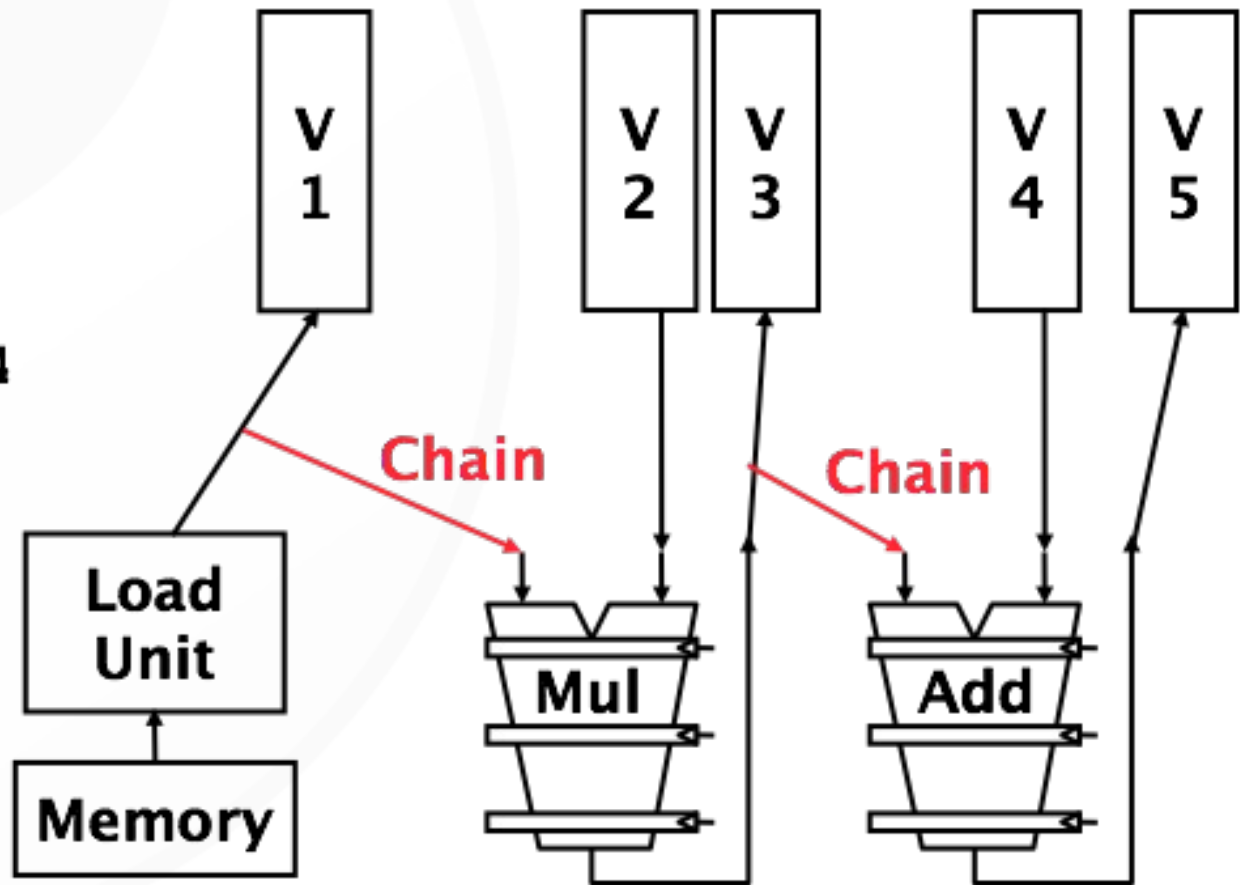
Chaining

- Sequences with read-after-write dependency hazards can be in the same convoy via chaining

- Chaining
  - Allows a vector operation to start as soon as the individual elements of its vector source operand become available

Vector Chaining

- Vector version of register bypassing
  - Introduced with Cray 1
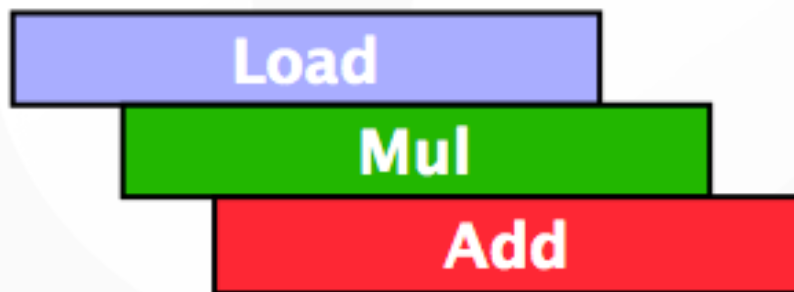
```
LV    v1
MULV  v3,v1,v2
ADDV  v5, v3, v4
```
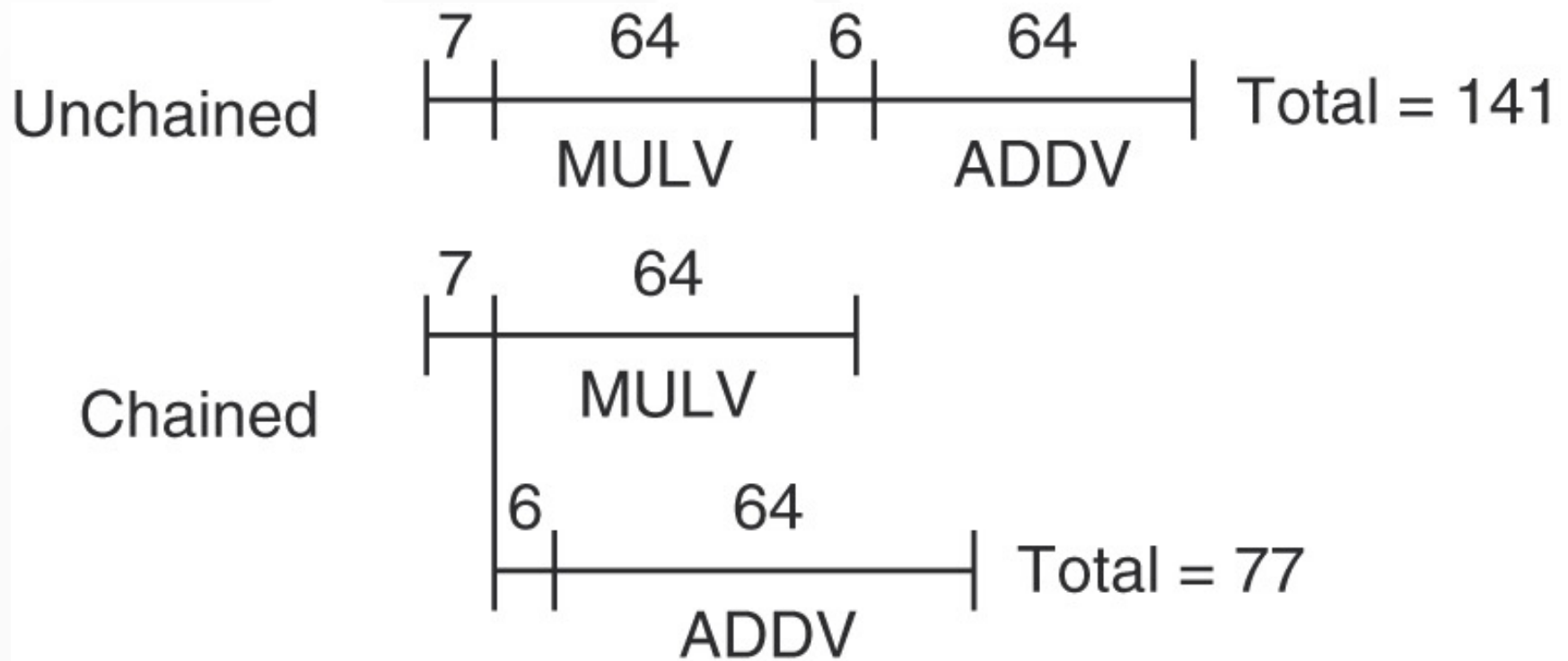
Vector Chaining

- Without chaining, must wait for last element of result to be written before starting dependent instruction

- With chaining, can start dependent instruction as soon as first result appears

Unchained vs. Chained

- Timing diagram for a sequence of dependent vector operations ADDV and MULV

Example

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

Convoys:

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

3 chimes, 2 FP ops per result, cycles per FLOP = 1.5

For 64 element vectors, requires 64 x 3 = 192 clock cycles

Convoy Time

- Show the time that each convoy can begin and the total # of cycles needed.
- Vector Start-Up Overhead

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

- Answer in terms of convoys, vector length n, and no convoy overlap

| 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	$30 + 3n$	$30 + 3n + 12$	$41 + 4n$

Convoy Time vs. Chime Approx

- How does the time compare to the chime approximation for a vector of length 64?
  - *Tricky Question: When is the vector sequence actually done?*
  - *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.*

Challenges

- Start-up time
  - Latency of vector functional unit
  - Assume the same as Cray-1
    - Floating-point add => 6 clock cycles
    - Floating-point multiply => 7 clock cycles
    - Floating-point divide => 20 clock cycles
    - Vector load => 12 clock cycles

- Improvemeants
  - > 1 element per clock cycle
  - Non-64 wide vectors
  - IF statements in vector code
  - Memory system optimizations to support vector processors
  - Multiple dimensional matrices
  - Sparse matrices
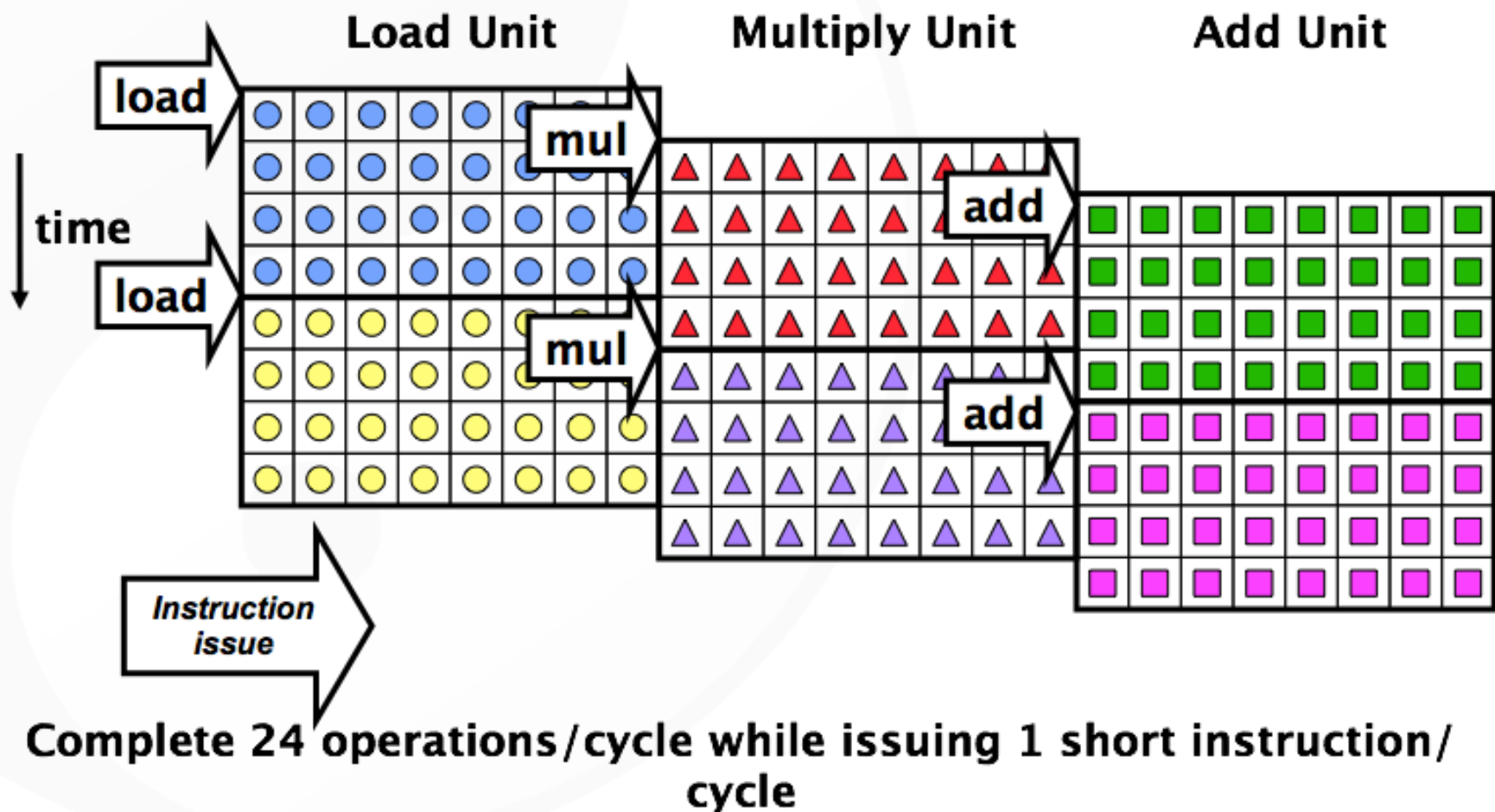 - Programming a vector computer

Multiple Lanes

- Element *n* of vector register *A* is "hardwired" to element *n* of vector register *B*
  - Allows for multiple hardware lanes
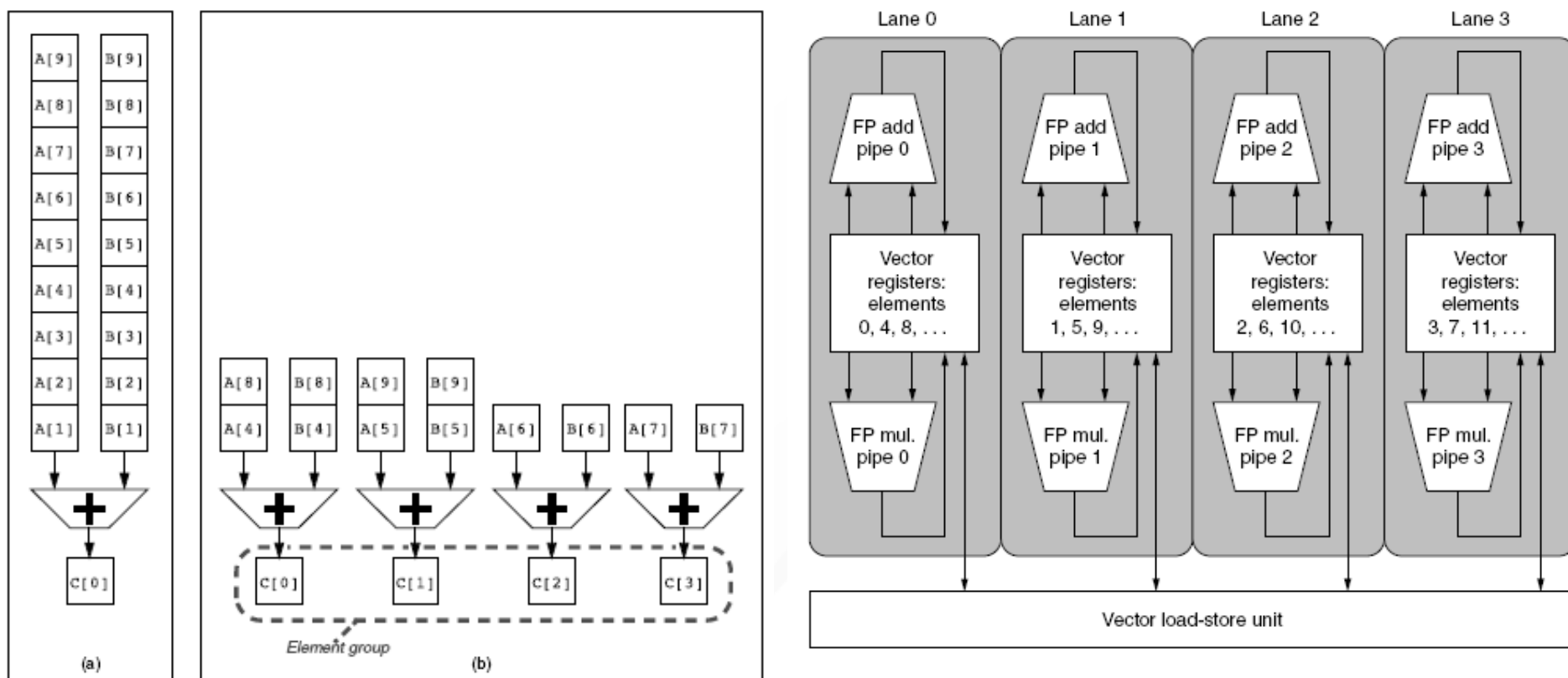
Advantages? Disadvantages?

Vector Instructions with Multiple Lanes and Chaining

- Can overlap execution of multiple vector instructions

Load Unit Multiply Unit Add Unit

load

mul

time

load

mul

add

add

Instruction issue

Complete 24 operations/cycle while issuing 1 short instruction/cycle

Vector Length Register

- Vector length not known at compile time?
- Use Vector Length Register (VLR) <= max vector length
- Use *strip mining* for vectors over the maximum length:

```
low = 0;
VL = (n % MVL);    /* find odd-size piece using modulo op % */
for (j = 0; j <= (n/MVL); j=j+1) {           /* outer loop */
   for (i = low; i < (low+VL); i=i+1) /* runs for length VL */
     Y[i] = a * X[i] + Y[i] ;              /* main operation */
   low = low + VL;                      /* start of next vector */
   VL = MVL;   /* reset the length to maximum vector length */
}
```
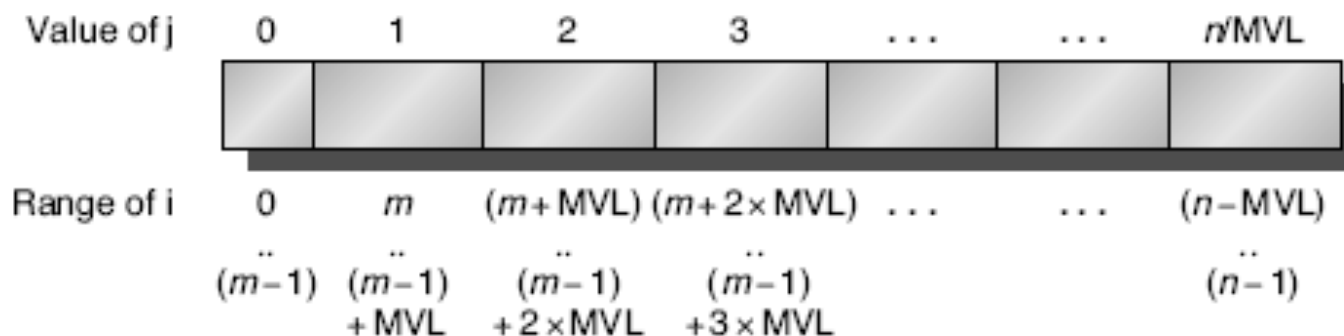
Vector Mask Registers

- Consider the following code snippet

```
for (i = 0; i < 64; i=i+1)
    if (X[i] != 0)
        X[i] = X[i] – Y[i];
```

This loop cannot *normally* be vectorized because of the conditional.

- Use vector mask register to "disable" elements

```
LV          V1,Rx           ;load vector X into V1
LV          V2,Ry           ;load vector Y
L.D         F0,#0           ;load FP zero into F0
SNEVS.D     V1,F0           ;sets VM(i) to 1 if V1(i)!=F0
SUBVV.D     V1,V1,V2        ;subtract under vector mask
SV          Rx,V1           ;store the result in X
```

- GFLOPS rate decreases!

Memory Banks

- Memory system must be designed to support high bandwidth for vector loads and stores

- Spread accesses across multiple banks
  - Control bank addresses independently
  - Load or store non-sequential words
  - Support multiple vector processors sharing the same memory

- Example:
  - 32 processors, each generating 4 loads and 2 stores/cycle
  - Processor cycle time is 2.167 ns, SRAM cycle time is 15 ns
  - How many memory banks needed?