

THREAD LEVEL PARALLELISM

The concept of thread

- smaller chunks of code (lightweight)
- threads are created within and belong to process
- for parallel thread processing, scheduling is performed on a per-thread basis
- less overhead on switching from thread to thread

Parallelism in Software

Instruction-level parallelism (ILP):

- Multiple instructions from the same instruction stream can be executed concurrently
- Generated and managed by hardware (superscalar) or by compiler (VLIW)
- Limited in practice by data and control dependences

Thread-level or task-level parallelism (TLP):

- Multiple threads or instruction sequences from the same application can be executed concurrently
- Generated by compiler/user and managed by compiler and hardware
- Limited in practice by communication/synchronization overheads and by algorithm characteristics

Parallelism in Software

Data-level parallelism (DLP):

- Instructions from a single stream operate concurrently on several data.
- Limited by non-regular data manipulation patterns and by memory bandwidth.

Transaction-level parallelism:

- Multiple threads/processes from different transactions can be executed concurrently.
- Limited by concurrency overheads.

Thread Level Parallel Architectures

- Architectures for exploiting thread-level parallelism

Hardware Multithreading

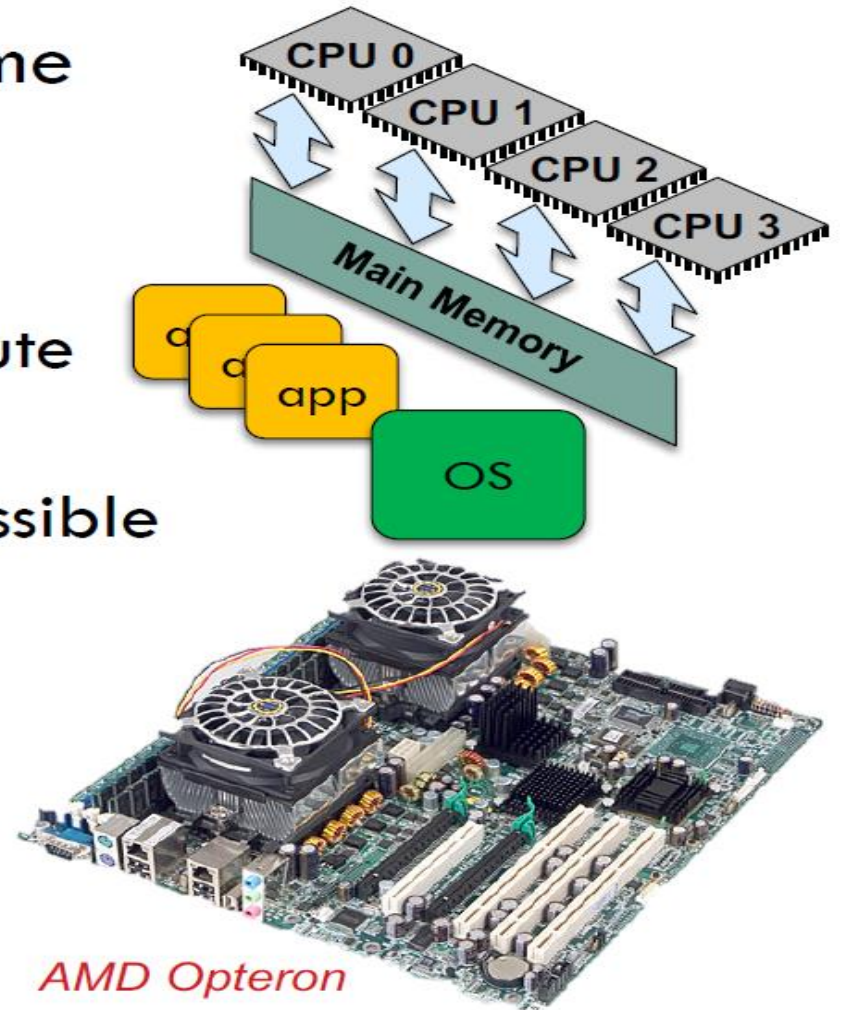
- Multiple threads run on the same processor pipeline
- Multithreading levels
 - Coarse grained multithreading (CGMT)
 - Fine grained multithreading (FGMT)
 - Simultaneous multithreading (SMT)

Multiprocessing

- Different threads run on different processors
- Two general types
 - Symmetric multiprocessors (SMP)
 - Single CPU per chip
 - Chip Multiprocessors (CMP)
 - Multiple CPUs per chip

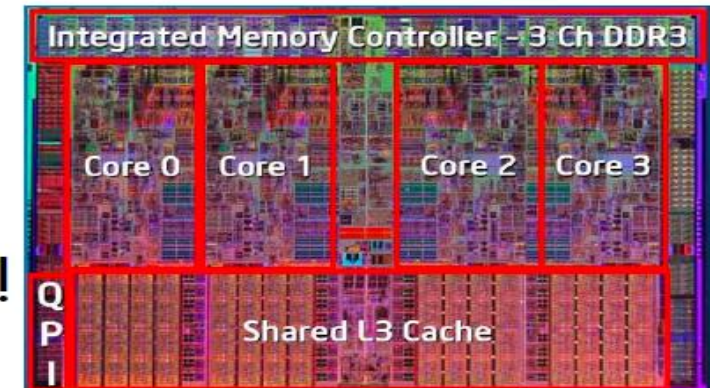
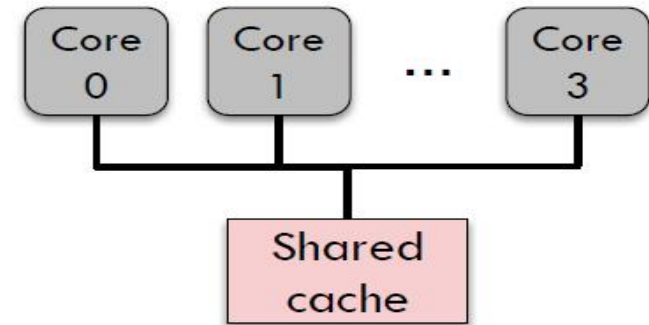
Symmetric Multiprocessors

- ❑ Multiple CPU chips share the same memory
- ❑ From the OS's point of view
 - ▣ All of the CPUs have equal compute capabilities
 - ▣ The main memory is equally accessible by the CPU chips
- ❑ OS runs every thread on a CPU
- ❑ Every CPU has its own power distribution and cooling system



Chip Multiprocessors

- ❑ Can be viewed as a simple SMP on single chip
- ❑ CPUs are now called cores
 - ▣ One thread per core
- ❑ Shared higher level caches
 - ▣ Typically the last level
 - ▣ Lower latency
 - ▣ Improved bandwidth
- ❑ Not necessarily homogenous cores!



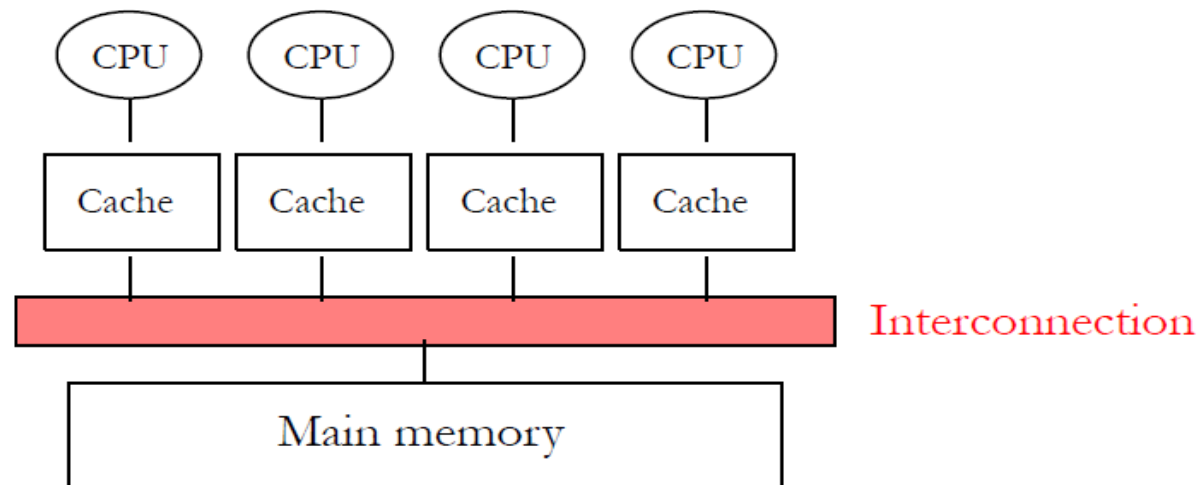
Intel Nehalem (Core i7)

Why Chip Multiprocessing?

- ❑ CMP exploits parallelism at lower costs than SMP
 - ▣ A single interface to the main memory
 - ▣ Only one CPU socket is required on the motherboard
- ❑ CMP requires less off-chip communication
 - ▣ Lower power and energy consumption
 - ▣ Better performance due to improved AMAT
- ❑ CMP better employs the additional transistors that are made available based on the Moore's law
 - ▣ More cores rather than more complicated pipelines

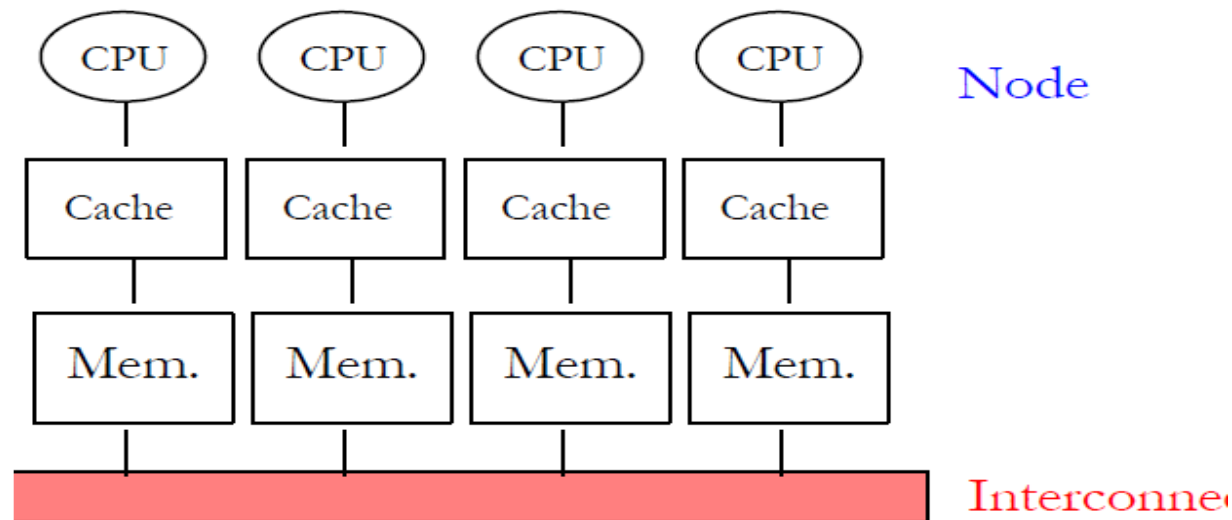
Taxonomy of Parallel Computers

- According to physical organization of processors and memory:
Physically centralized memory, **uniform memory access (UMA)**
- All memory is allocated at same distance from all processors
- Also called symmetric multiprocessors (SMP)
- Memory bandwidth is fixed and must accommodate all processors → does not scale to large number of processors.



Taxonomy of Parallel Computers

- According to physical organization of processors and memory:
Physically distributed memory, **non-uniform memory access (NUMA)**
- A portion of memory is allocated with each processor (**node**)
- Accessing **local** memory is much faster than **remote** memory
- If most accesses are to local memory than overall memory bandwidth increases linearly with the number of processors.



Taxonomy of Parallel Computers

According to memory communication model

Shared address or **shared memory**

- Processes in different processors can use the same virtual address space
- Any processor can directly access memory in another processor node
- Communication is done through shared memory variables
- Explicit synchronization with locks and critical sections

Distributed address or **message passing**

- Processes in different processors use different virtual address spaces
- Each processor can only directly access memory in its own node
- Communication is done through explicit messages
- Synchronization is implicit in the messages
- Some standard message passing libraries (e.g., MPI)

Shared Memory vs. Message Passing


- Shared memory

Producer (p1)

```
flag = 0;  
...  
a = 10;  
flag = 1;
```

Consumer (p2)

```
flag = 0;  
...  
while (!flag) {}  
x = a * y;
```




- Message passing

Producer (p1)

```
...  
a = 10;  
send(p2, a, label);
```

Consumer (p2)

```
...  
receive(p1, b, label);  
x = b * y;
```



Vectorization

Vectorization is the process of converting an algorithm from operating on a single value at a time to operating on a set of values (vector) at one time.

Modern CPUs provide direct support for vector operations where a single instruction is applied to multiple data (SIMD).

For example, a CPU with a 512 bit register could hold 16 32- bit single precision doubles and do a single calculation.

16 times faster than executing a single instruction at a time. Combine this with threading and multi-core CPUs leads to orders of magnitude performance gains.

Vector is alternative model for exploiting ILP .

Vector Operations

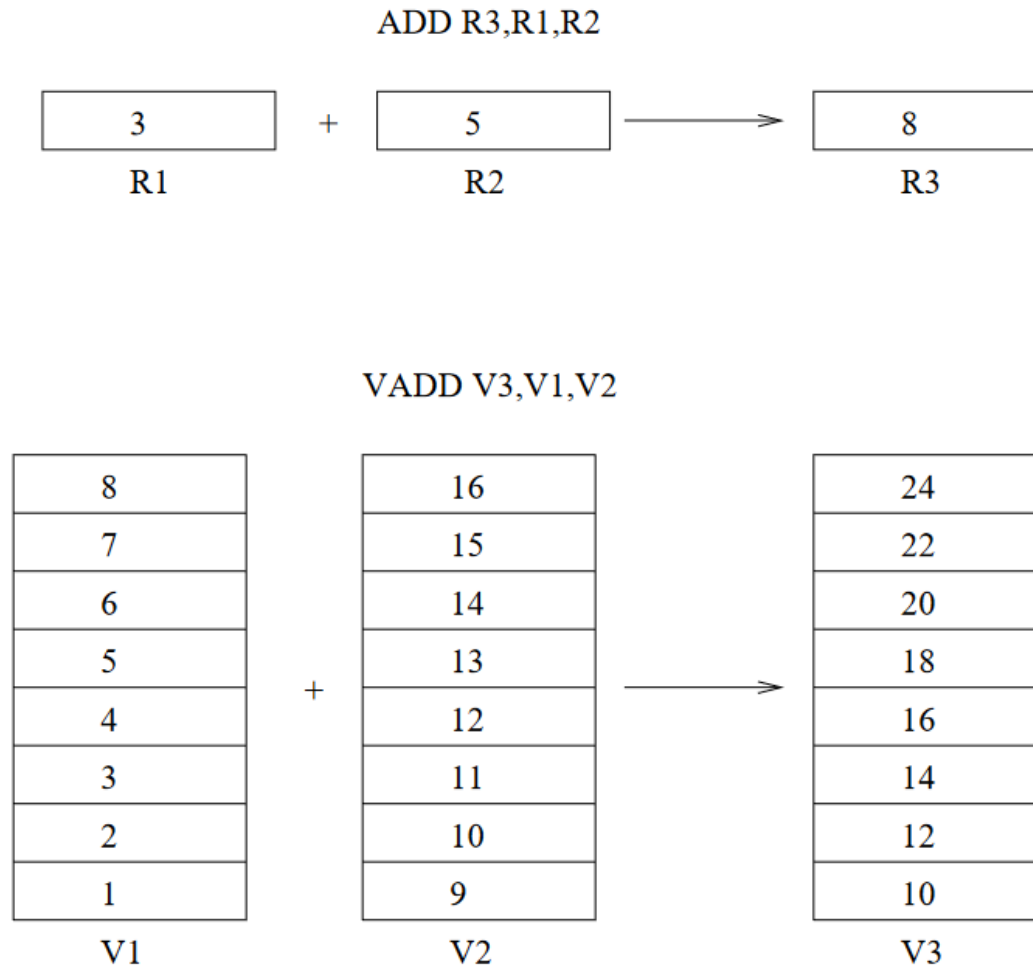


Figure 2: Difference between scalar and vector add instructions

Vector Code Example:

```
# C code
for (i=0; i<64; i++)
    C[i] = A[i] + B[i];
```

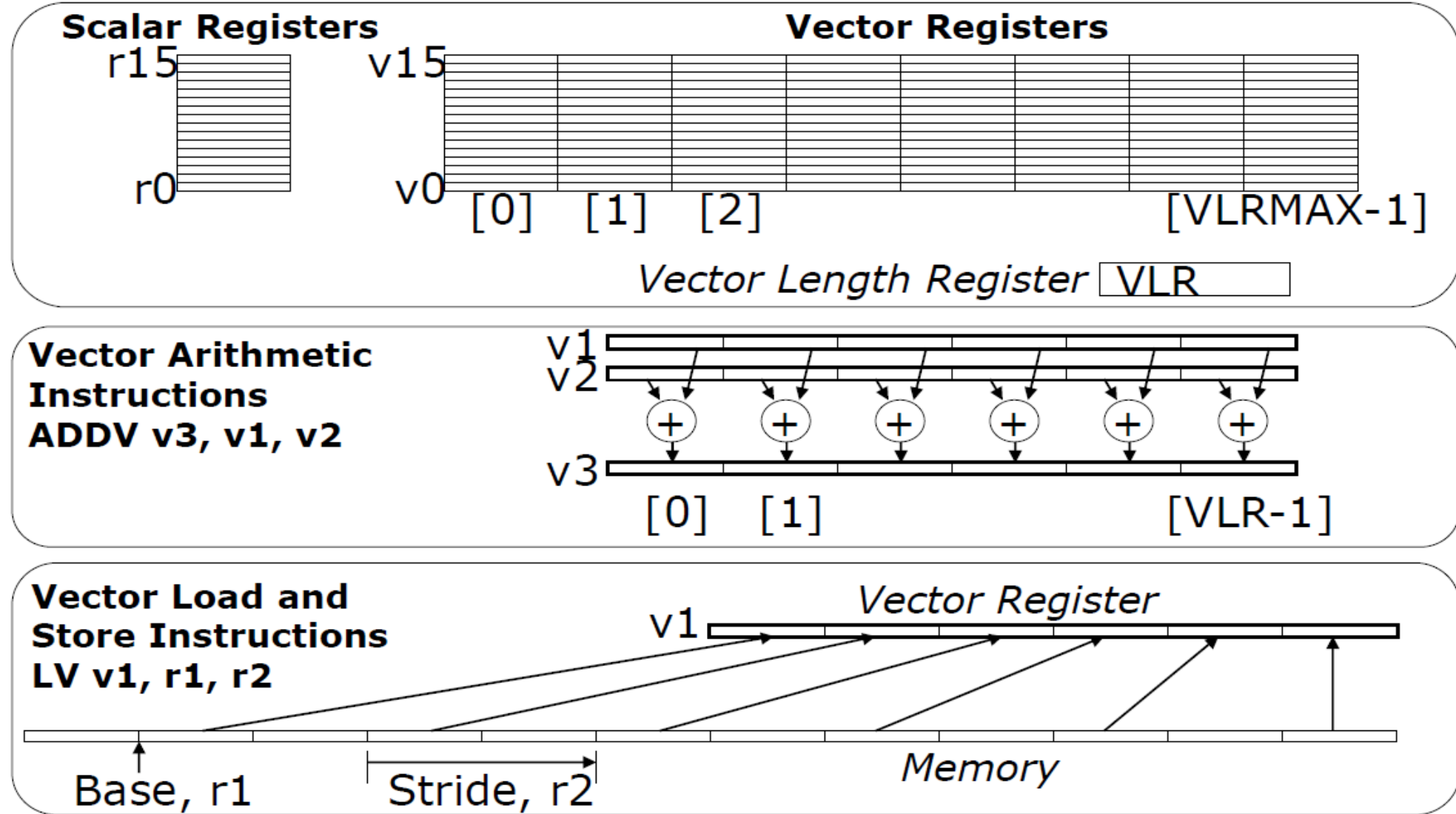
```
# Scalar Code
LI R4, 64
loop:
    L.D F0, 0(R1)
    L.D F2, 0(R2)
    ADD.D F4, F2, F0
    S.D F4, 0(R3)
    DADDIU R1, 8
    DADDIU R2, 8
    DADDIU R3, 8
    DSUBIU R4, 1
    BNEZ R4, loop
```

```
# Vector Code
LI VLR, 64
LV V1, R1
LV V2, R2
ADDV.D V3, V1, V2
SV V3, R3
```

Load immediate (LI) with length of vector (64)

Vector length register (VLR)

Vector Programming Model



Stride is the distance separating elements in memory that will be adjacent in a *vector register*

Vector Instruction Set Advantages

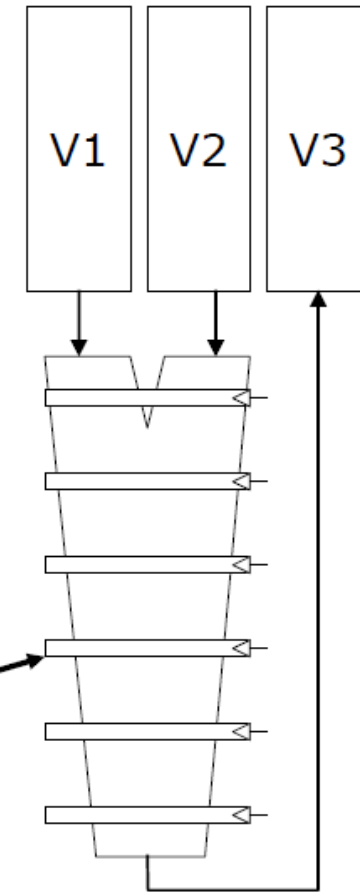
- Compact
 - one short instruction encodes N operations
- Expressive, tells hardware that these N operations:
 - are independent
 - use the same functional unit
 - access disjoint registers
 - access registers in the same pattern as previous instructions
 - access a contiguous block of memory (unit-stride load/store)
 - access memory in a known pattern (strided load/store)
- Scalable
 - can run same object code on more parallel pipelines/lane

Vector Arithmetic Execution

Use deep pipeline (fast clock)
to execute operations for
each vector element.

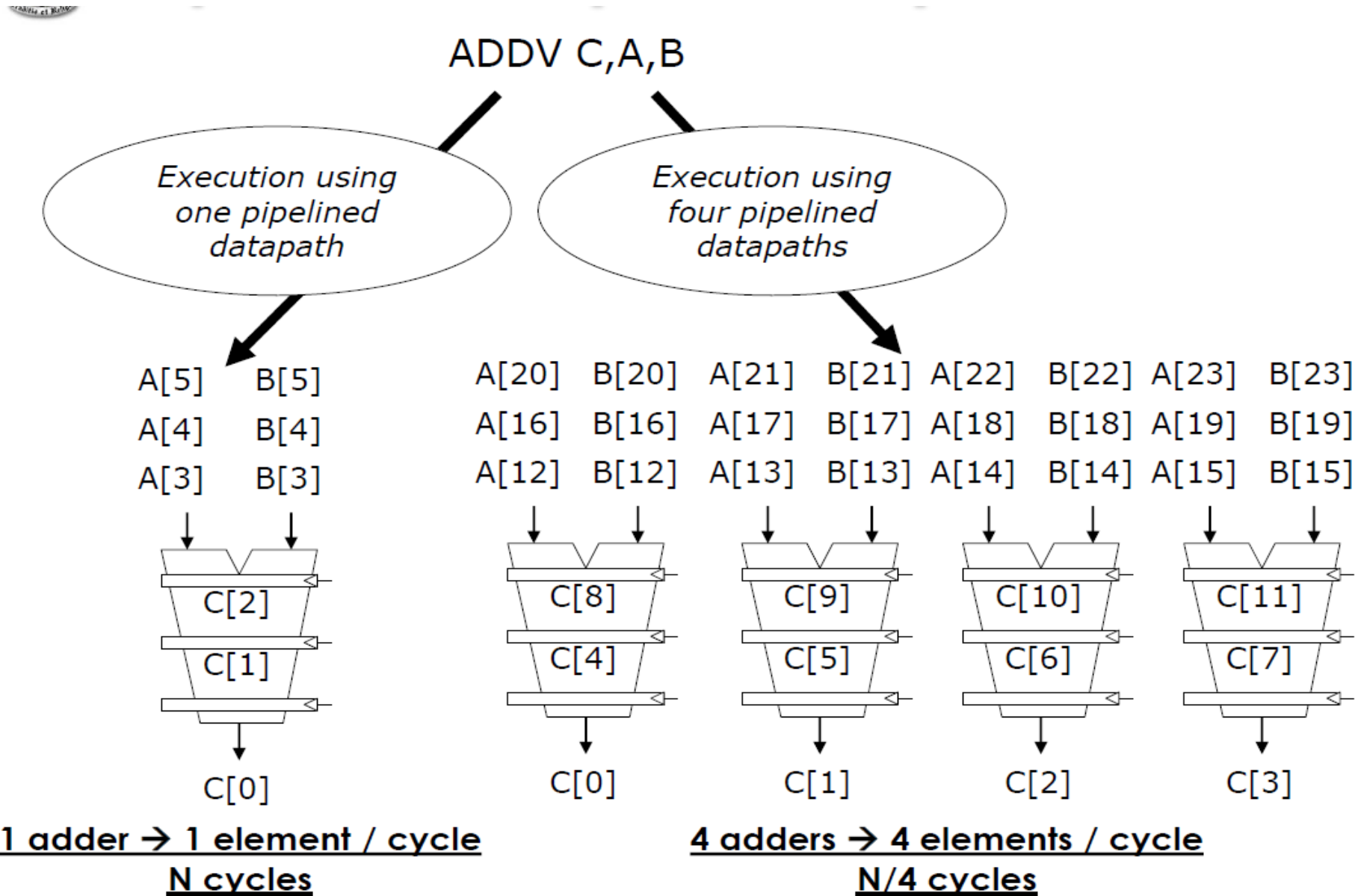
Simplify pipeline control
because elements in vector
are independent → no
hazards.

Six stage multiply pipeline

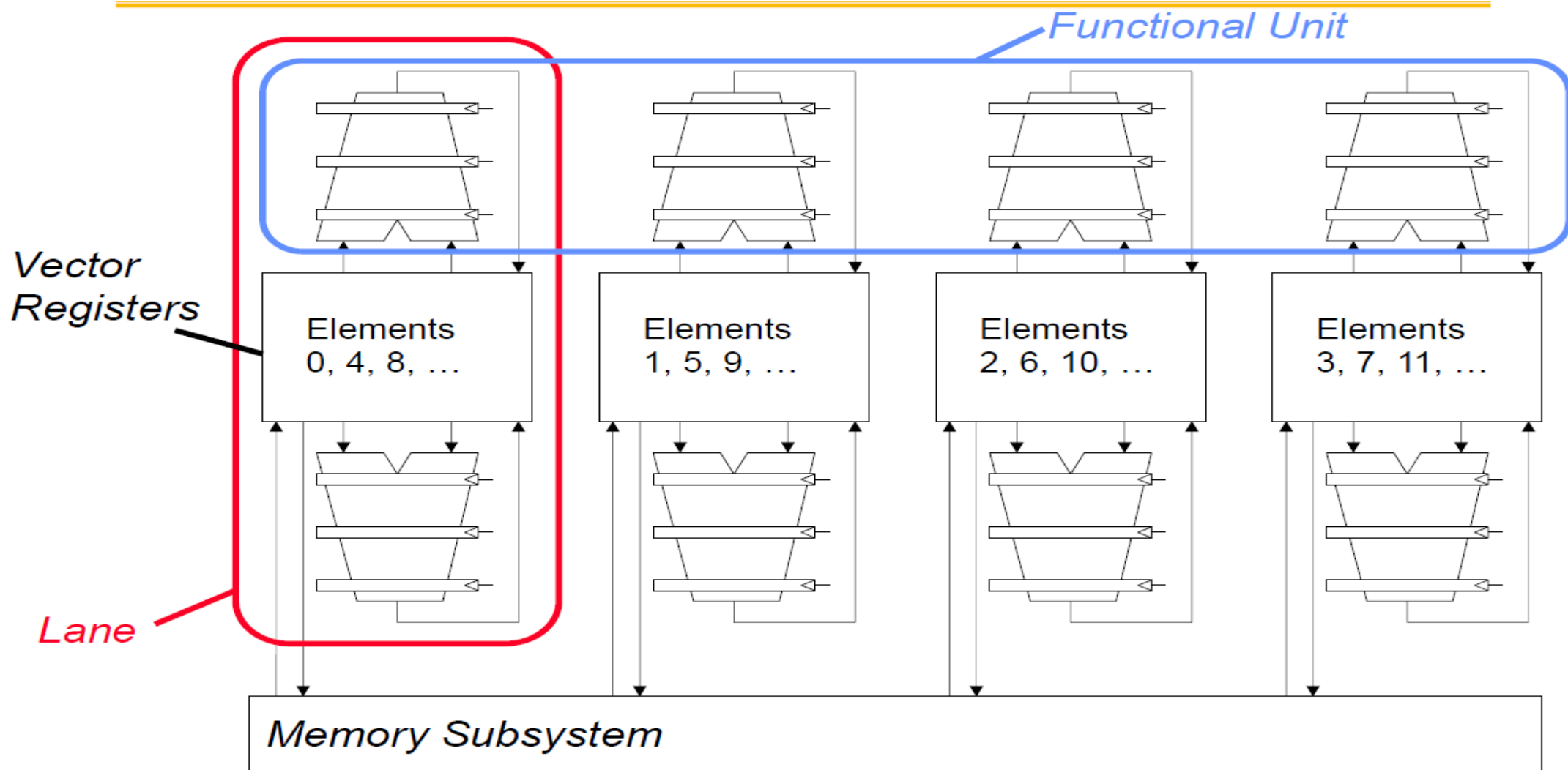


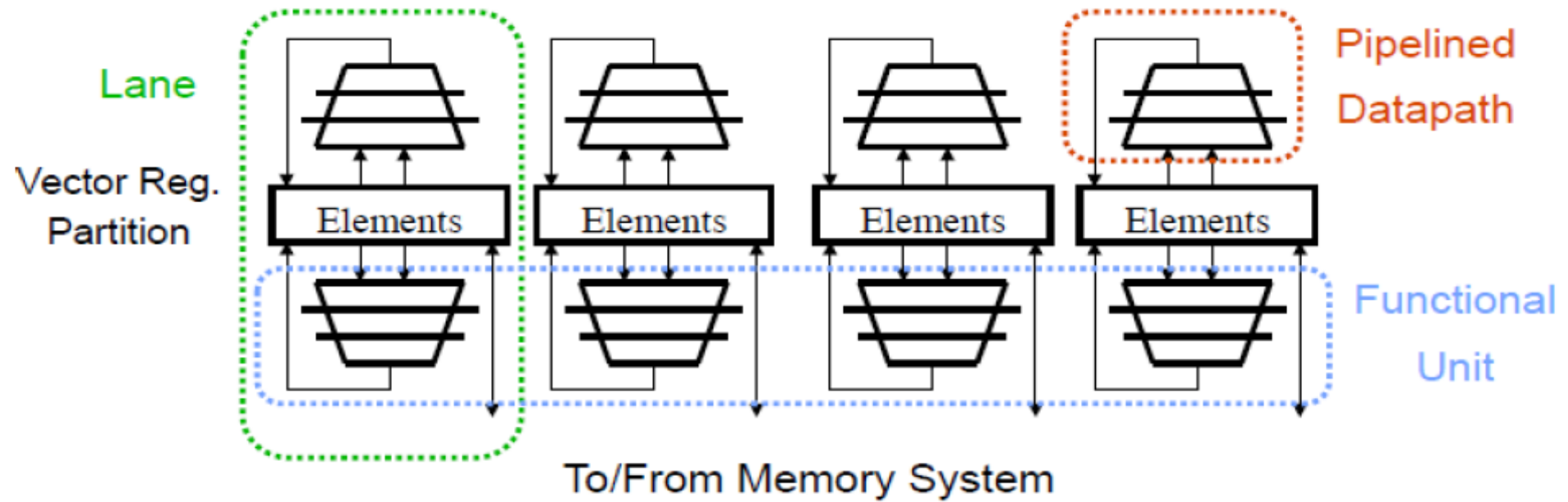
$$V3[i] \leftarrow V1[i] * V2[i]$$

Multiple Datapath



Vector Unit Structure



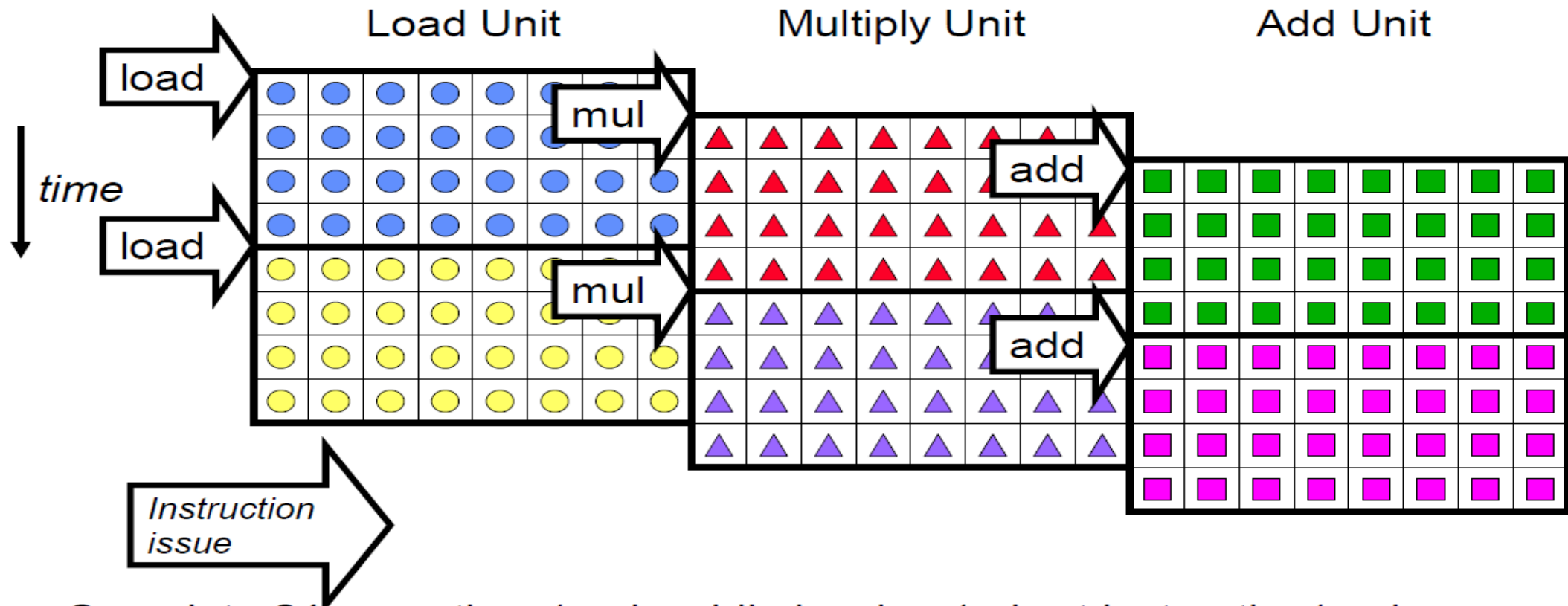


- Vector elements interleaved across lanes
- Example: $V[0, 4, 8, \dots]$ on Lane 1, $V[1, 5, 9, \dots]$ on Lane 2, etc.
- Compute for multiple elements per cycle
- Example: Lane 1 computes on $V[0]$ and $V[4]$ in one cycle
- Modular, scalable design
- No inter-lane communication needed for most vector instructions

Vector Instruction Parallelism

Can overlap execution of multiple vector instructions

- example machine has 32 elements per vector register and 8 lanes



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