

# CSCI 564 Advanced Computer Architecture

## Lecture 12: Vector Processing and SIMD

---

Dr. Bo Wu

April 23, 2021

Colorado School of Mines

# Flynn's Taxonomy of Computers

- Mike Flynn, “[Very High-Speed Computing Systems,](#)” Proc. of IEEE, 1966
- **SISD**: Single instruction operates on single data element
- **SIMD**: Single instruction operates on multiple data elements
  - Array processor
  - Vector processor
- **MISD**: Multiple instructions operate on single data element
- **MIMD**: Multiple instructions operate on multiple data elements (multiple instruction streams)
  - Multiprocessor
  - Multithreaded processor

# Data Parallelism

- Concurrency arises from performing the **same operations on different pieces of data**
  - Single instruction multiple data (SIMD)
  - E.g., dot product of two vectors
- Contrast with data flow
  - Concurrency arises from executing different operations in parallel (in a data driven manner)

# SIMD Processing

- Single instruction operates on multiple data elements
  - In time or in space
- Multiple processing elements
- Time-space duality
  - **Array processor**: Instruction operates on multiple data elements at the same time
  - **Vector processor**: Instruction operates on multiple data elements in consecutive time steps

# Array vs. Vector Processors

ARRAY PROCESSOR



Instruction Stream

LD VR ← A[3:0]  
ADD VR ← VR, 1  
MUL VR ← VR, 2  
ST A[3:0] ← VR

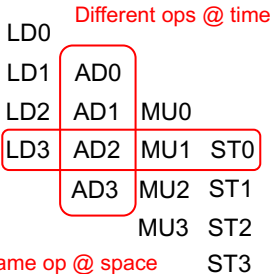
Time



Different ops @ same space

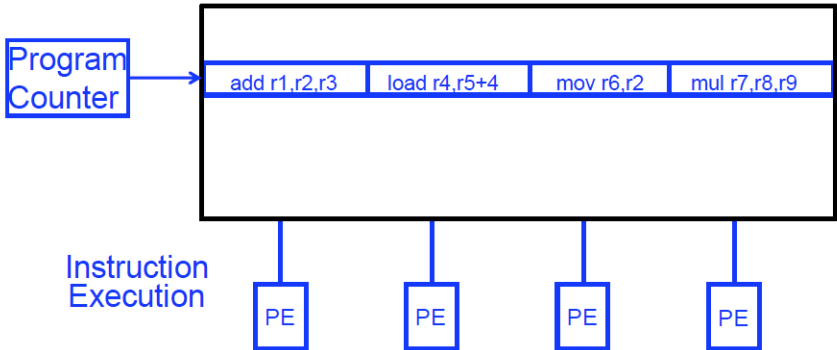
Space

VECTOR PROCESSOR

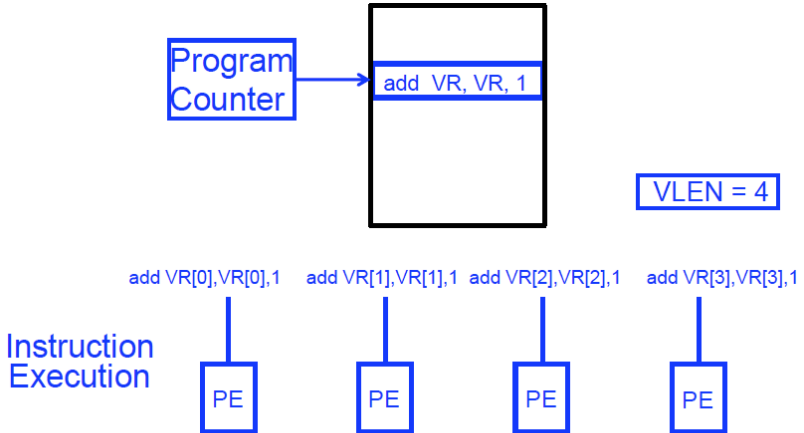


Space

# SIMD Array Processing vs. VLIW



# SIMD Array Processing vs. VLIW



# Vector Processors

- A vector is a one-dimensional array of numbers
- Many scientific/commercial programs use vectors

```
for (i = 0; i<=49; i++)  
    C[i] = (A[i] + B[i]) / 2
```

- A vector processor is one whose instructions operate on vectors rather than scalar (single data) values
- Basic requirements
  - Need to load/store vectors → vector registers (contain vectors)
  - Need to operate on vectors of different lengths → vector length register (VLEN)
  - Elements of a vector might be stored apart from each other in memory → vector stride register (VSTR)
    - Stride: distance between two elements of a vector



# Vector Processors (II)

- A vector instruction performs an operation on each element in consecutive cycles
  - Vector functional units are pipelined
  - Each pipeline stage operates on a different data element
- Vector instructions allow deeper pipelines
  - No intra-vector dependencies
  - No control flow within a vector
  - Known stride allows prefetching of vectors into cache/memory

# Vector Processor Advantages

- + No dependencies within a vector
  - Pipelining, parallelization work well
  - Can have very deep pipelines, no dependencies!
- + Each instruction generates a lot of work
  - Reduces instruction fetch bandwidth
- + Highly regular memory access pattern
  - Interleaving multiple banks for higher memory bandwidth
  - Prefetching
- + No need to explicitly code loops
  - Fewer branches in the instruction sequence

# Vector Processor Disadvantages

- Works (only) if parallelism is regular (data/SIMD parallelism)
  - ++ Vector operations
  - Very inefficient if parallelism is irregular
    - How about searching for a key in a linked list?

**To program a vector machine, the compiler or hand coder must make the data structures in the code fit nearly exactly the regular structure built into the hardware. That's hard to do in first place, and just as hard to change. One tweak, and the low-level code has to be rewritten by a very smart and dedicated programmer who knows the hardware and often the subtleties of the application area. Often the rewriting is**

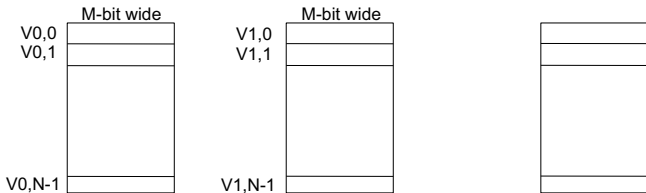
# Vector Processor Limitations

- Memory (bandwidth) can easily become a bottleneck, especially if
  1. compute/memory operation balance is not maintained
  2. data is not mapped appropriately to memory banks

# Vector Processing in More Depth

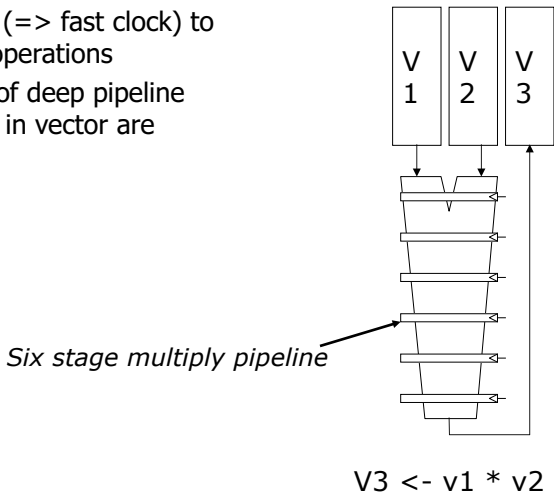
# Vector Registers

- Each **vector data register** holds N M-bit values
- **Vector control registers**: VLEN, VSTR, VMASK
- Maximum VLEN can be N
  - Maximum number of elements stored in a vector register
- **Vector Mask Register (VMASK)**
  - Indicates which elements of vector to operate on
  - Set by vector test instructions
    - e.g.,  $VMASK[i] = (V_k[i] == 0)$

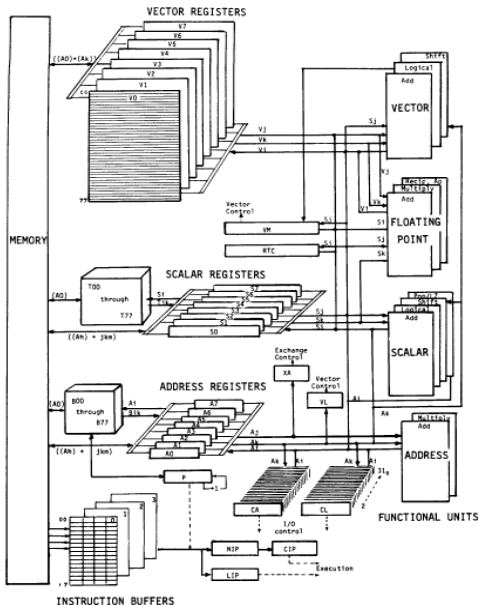


# Vector Functional Units

- Use deep pipeline ( $\Rightarrow$  fast clock) to execute element operations
- Simplifies control of deep pipeline because elements in vector are independent



# Vector Machine Organization (CRAY-1)

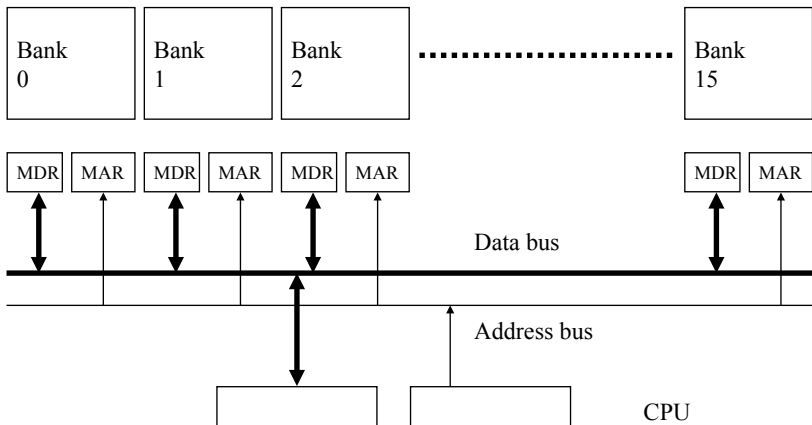


- CRAY-1
- Russell, “The CRAY-1 computer system,” CACM 1978.
- Scalar and vector modes
- 8 64-element vector registers
- 64 bits per element
- 16 memory banks
- 8 64-bit scalar registers

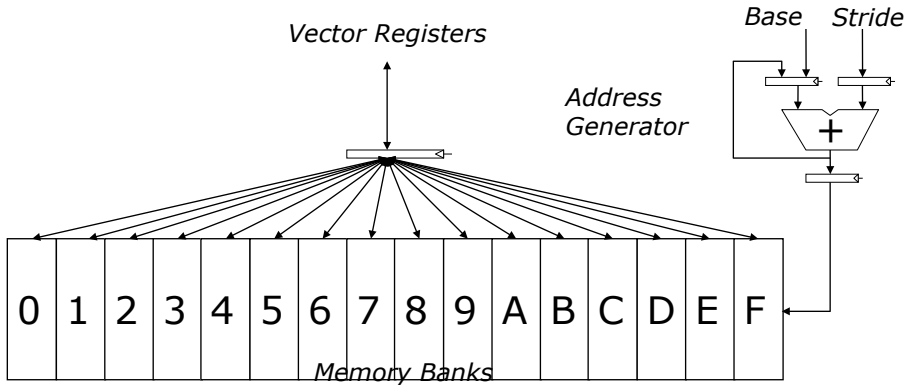


# Memory Banking

- Memory is divided into banks that can be accessed independently; banks share address and data buses
- Can start and complete one bank access per cycle
- Can sustain N parallel accesses if they go to different banks



# Vector Memory System



# Scalar Code Example

- For I = 0 to 49
  - $C[i] = (A[i] + B[i]) / 2$
- Scalar code (instruction and its latency)

MOVI R0 = 50            1

MOVA R1 = A            1

MOVA R2 = B            1

MOVA R3 = C            1

304 dynamic instructions

X: LD R4 = MEM[R1++]        11 ;autoincrement addressing

LD R5 = MEM[R2++]        11

ADD R6 = R4 + R5        4

SHFR R7 = R6 >> 1        1

ST MEM[R3++] = R7        11

DECBNZ --R0, X        2 ;decrement and branch if NZ

# Scalar Code Execution Time

- Scalar execution time on an in-order processor with 1 bank
  - First two loads in the loop cannot be pipelined:  $2 \times 11$  cycles
  - $4 + 50 \times 40 = 2004$  cycles
- Scalar execution time on an in-order processor with 16 banks (word-interleaved: consecutive words are stored in consecutive banks)
  - First two loads in the loop can be pipelined
  - $4 + 50 \times 30 = 1504$  cycles
- Why 16 banks?
  - 11 cycle memory access latency
  - Having 16 ( $>11$ ) banks ensures there are enough banks to overlap enough memory operations to cover memory latency

# Vectorizable Loops

- A loop is **vectorizable** if each iteration is independent of any other
- For  $I = 0$  to 49
  - $C[i] = (A[i] + B[i]) / 2$  7 dynamic instructions

- Vectorized loop:

MOVI VLEN = 50	1
MOVI VSTR = 1	1
VLD V0 = A	$11 + VLN - 1$
VLD V1 = B	$11 + VLN - 1$
VADD V2 = V0 + V1	$4 + VLN - 1$
VSHFR V3 = V2 >> 1	$1 + VLN - 1$
VST C = V3	$11 + VLN - 1$

# Conditional Operations in a Loop

- What if some operations should not be executed on a vector (based on a dynamically-determined condition)?

```
loop:   if (a[i] != 0) then b[i]=a[i]*b[i]
        goto loop
```

- Idea: **Masked operations**

- VMASK register is a bit mask determining which data element should not be acted upon

```
VLD V0 = A
```

```
VLD V1 = B
```

```
VMASK = (V0 != 0)
```

```
VMUL V1 = V0 * V1
```

```
VST B = V1
```

- Does this look familiar? This is essentially **predicated execution**.

# Another Example with Masking

```
for (i = 0; i < 64; ++i)
  if (a[i] >= b[i]) then c[i] = a[i]
  else c[i] = b[i]
```

A	B	VMASK
1	2	0
2	2	1
3	2	1
4	10	0
-5	-4	0
0	-3	1
6	5	1
-7	-8	1

Steps to execute loop

1. Compare A, B to get VMASK
2. Masked store of A into C
3. Complement VMASK
4. Masked store of B into C

# Masked Vector Instructions

## Simple Implementation

- execute all N operations, turn off result writeback according to mask

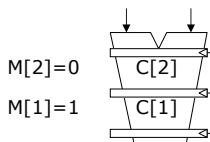
M[7]=1 A[7] B[7]

M[6]=0 A[6] B[6]

M[5]=1 A[5] B[5]

M[4]=1 A[4] B[4]

M[3]=0 A[3] B[3]



M[2]=0

M[1]=1

M[0]=0

Write Enable

C[0]

Write data port

## Density-Time Implementation

- scan mask vector and only execute elements with non-zero masks

M[7]=1

M[6]=0

M[5]=1

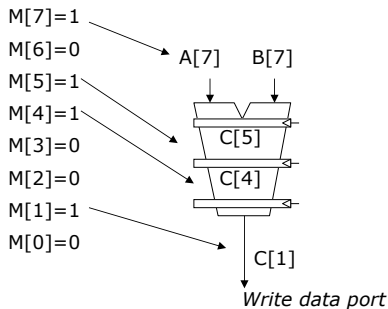
M[4]=1

M[3]=0

M[2]=0

M[1]=1

M[0]=0



A[7] B[7]

C[5]

C[4]

C[1]

Write data port



# Vector/SIMD Processing Summary

- Vector/SIMD machines are good at exploiting **regular data-level parallelism**
  - Same operation performed on many data elements
  - Improve performance, simplify design (no intra-vector dependencies)
- **Performance improvement limited by vectorizability** of code
  - Scalar operations limit vector machine performance
  - Amdahl's Law
  - CRAY-1 was the fastest SCALAR machine at its time!
- Many existing ISAs include (vector-like) SIMD operations
  - Intel MMX/SSEn/AVX, PowerPC AltiVec, ARM Advanced SIMD