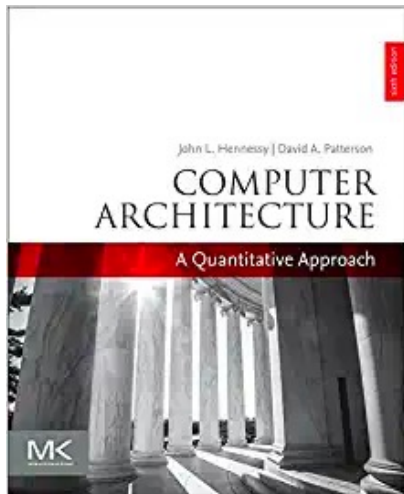


Chapter 4

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

Part 1: Introduction & Vector Architectures



“Who’s first?”

“America.”

“Who’s second?”

“Sir, there is no second.”

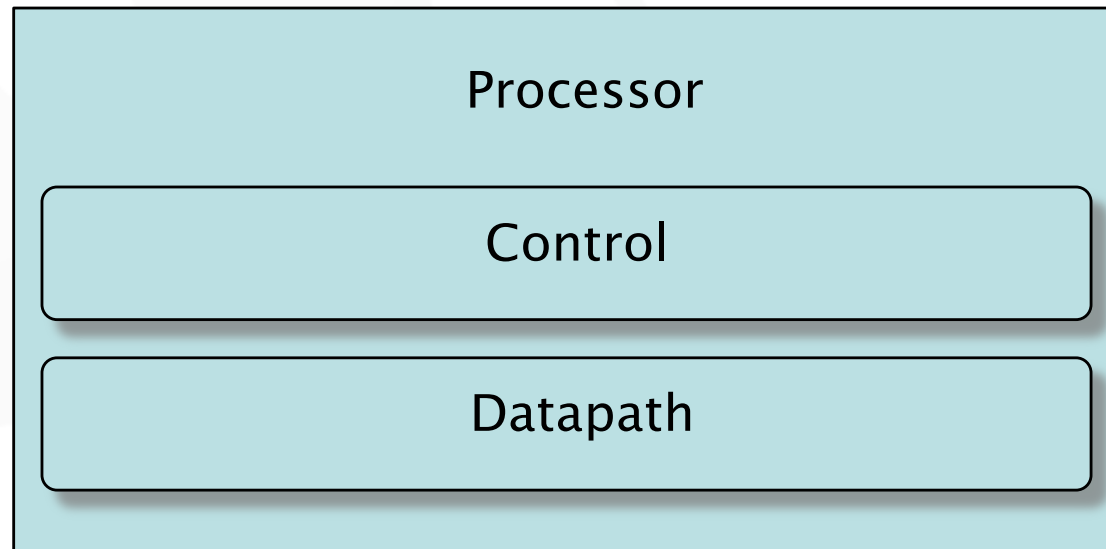
-Dialog between two observers of the sailing race later named “The America’s Cup” and run every few years -- the inspiration for John Cocke’s naming of the IBM research processor as “America.” This processor was the precursor to the RS/6000 series and the first superscalar microprocessor.

Acknowledgements

- Thanks to many sources for slide material

- © 1990 Morgan Kaufmann Publishers, © 2001-present Elsevier
Computer Architecture: A Quantitative Approach by J. Hennessy & D. Patterson
- © 1994 Morgan Kaufmann Publishers, © 2001-present Elsevier
Computer Organization and Design by D. Patterson & J. Hennessy
- © 2002 K. Asinovic & Arvind, MIT
- © 2002 J. Kubiawicz, University of California at Berkeley
- © 2006, © 2010 No Starch Press for Inside the Machine by J. Stokes
- © 2007 W.-M. Hwu & D. Kirk, University of Illinois & NVIDIA
- © 2007-2010 J. Owens, University of California at Davis
- © 2010 CRC Press for Introduction to Concurrency in Programming Languages by M. Sottile, T. Mattson, and C. Rasmussen
- © 2017, IBM POWER9 Processor Architecture by Sadasivam et al., IBM
- © 2016, © 2019 POWER9 Processor User's Manual, IBM
- © The OpenPOWER Foundation
- © 2022, W. Feng, Virginia Tech

What We Know



- What new techniques have we learned that make ...
 - ... *control* go fast?
 - ... *datapath* go fast?

Flynn's Classification Scheme



- SISD – single instruction, single data stream
 - uniprocessors
- SIMD – single instruction, multiple data streams
 - single control unit broadcasting operations to multiple datapaths
- MISD – multiple instruction, single data stream
 - no such machine
(though some people put vector machines in this category)
- MIMD – multiple instructions, multiple data streams
 - *a.k.a* multiprocessors (SMPs, MPPs, clusters, NOWs)

Continuum of Granularity

- “Coarse”
 - Each processor is more powerful
 - Usually fewer processors
 - Communication is more expensive between processors
 - Processors are more loosely coupled
 - Tend toward MIMD
- “Fine”
 - Each processor is less powerful
 - Usually more processors
 - Communication is cheaper between processors
 - Processors are more tightly coupled
 - Tend toward SIMD

“If you were plowing a field, which would you rather use?
Two strong oxen or 1024 chickens?”

- Seymour Cray

The Road Ahead

- What's Up?
 - Chapter 4: Data-level parallelism (DLP). Fine-grained parallelism.
 - Vector & stream processors
 - SIMD extensions: MMX → SSE
 - Graphics processing units (GPUs)
- What We Just Completed
 - Chapter 5: Thread-level parallelism (TLP). Coarse-grained parallelism.
 - Multicore
 - Multiprocessors
 - Clusters

Introduction to SIMD

- SIMD architectures can exploit significant data-level parallelism for
 - Matrix-oriented scientific computing
 - Media-oriented image and sound processors
- SIMD is more energy efficient than MIMD
 - Only needs to fetch one instruction per data operation
 - Makes SIMD attractive for personal mobile devices
- SIMD allows programmer to continue to think sequentially

SIMD Parallelism

- Vector architectures (VMIPS, Motorola AltiVec)
- SIMD extensions (MMX → SSE → AVX)
- Graphics processing units (GPUs)
- For x86 processors
 - Expect two additional cores per chip per year
 - Expect SIMD width to double every four years
 - Expect potential speedup from SIMD to be twice that from MIMD

SIMD Parallelism

- Vector architectures (VMIPS, Motorola AltiVec)
- SIMD extensions (MMX → SSE → AVX)
- Graphics processing units (GPUs)
- For x86 processors
 - Expect two additional cores per chip per year
 - Expect SIMD width to double every four years
 - Expect potential speedup from SIMD to be twice that from MIMD

Seymour Cray

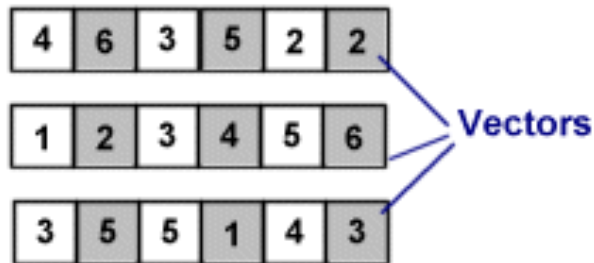
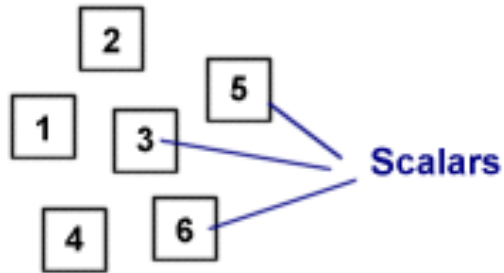
The Father of Supercomputing



An American electrical engineer and supercomputer architect who designed a series of computers that were the fastest in the world for decades..

- “Anyone can build a fast CPU. The trick is to build a fast system.”
- When asked what kind of CAD tools he used for the Cray-1, Cray said that he liked “#3 pencils with quadrille pads.”
- When he was told that Apple Computer had just bought a Cray to help design the next Apple Macintosh, Cray commented that he had just bought a Macintosh to design the next Cray.
- In 70s–80s, Supercomputer \equiv Vector Machine

Scalar vs. Vector



Representing Vectors

- Multiple items within same data word
- Multiple data words

- “The basic unit of SIMD is the vector, which is why SIMD computing is also known as vector processing. A vector is nothing more than a row of individual numbers or scalars.”

Vector Code Example

<pre># C code for (i=0; i<64; i++) C[i] = A[i] + B[i];</pre>	<pre># 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</pre>	<pre># Vector Code LI VLR, 64 LV V1, R1 LV V2, R2 ADDV.D V3, V1, V2 SV V3, R3</pre>
---	---	---

- What are the advantages of vector code?

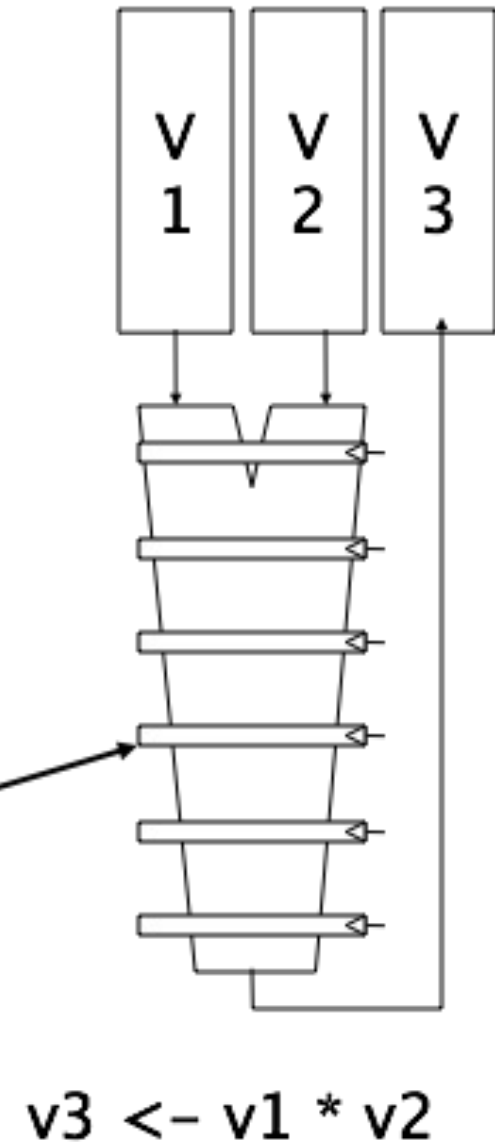
Vector Instruction Set

- 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), or
 - access memory in a known pattern (strided load/store)
- Scalable
 - can run same object code on more parallel pipelines or lanes

Vector Arithmetic

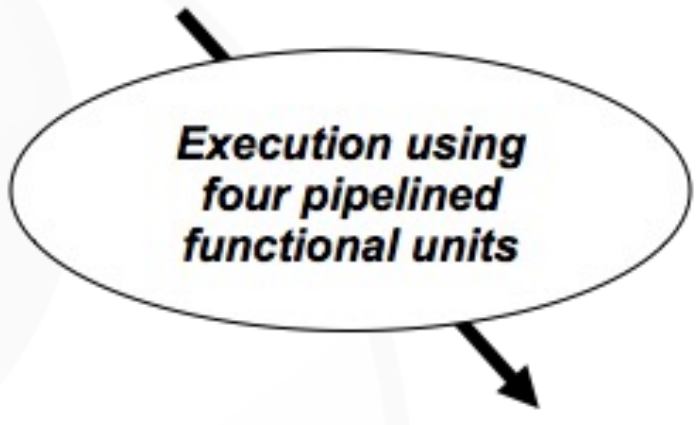
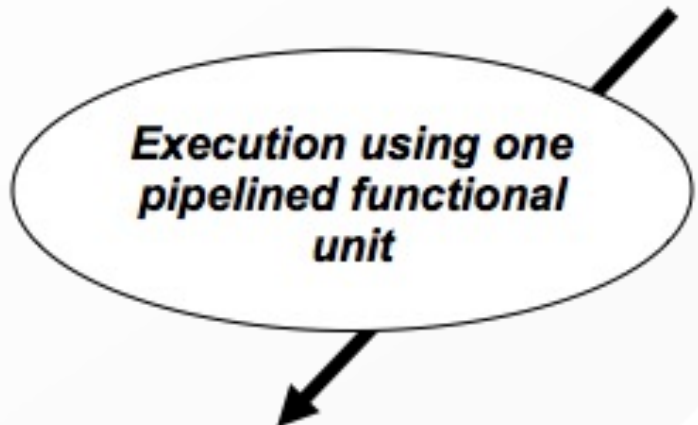
- Use deep pipeline (\Rightarrow fast clock) to execute element operations
- Simplifies control of deep pipeline because elements in vector are independent (\Rightarrow no hazards!)

Six stage multiply pipeline



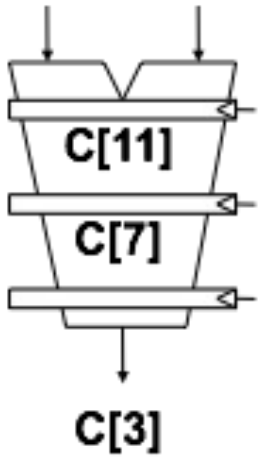
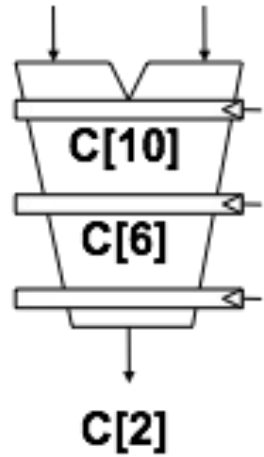
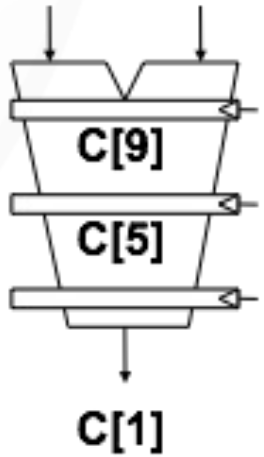
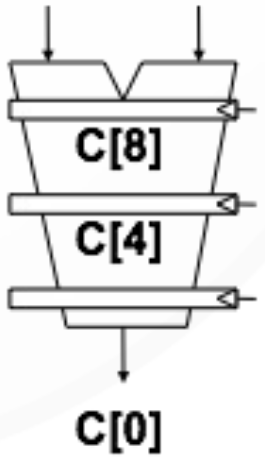
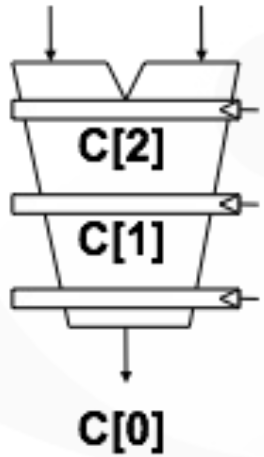
Vector Instruction

ADDV C, A, B



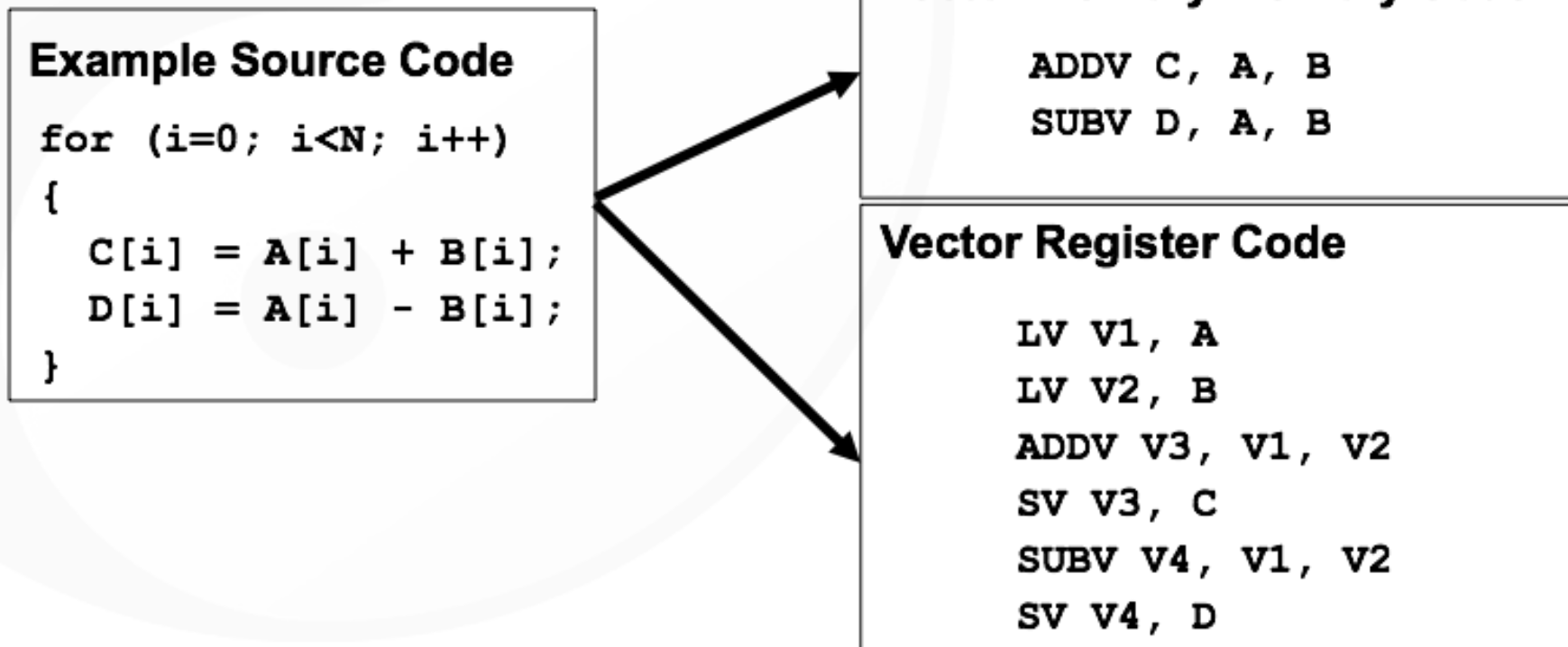
A[6] B[6]
 A[5] B[5]
 A[4] B[4]
 A[3] B[3]

A[24] B[24] A[25] B[25] A[26] B[26] A[27] B[27]
 A[20] B[20] A[21] B[21] A[22] B[22] A[23] B[23]
 A[16] B[16] A[17] B[17] A[18] B[18] A[19] B[19]
 A[12] B[12] A[13] B[13] A[14] B[14] A[15] B[15]



Vector Memory-Memory vs. Vector Register Machines

- Vector memory-memory instructions hold all vector operands in main memory
- The first vector machines, CDC Star-100 ('73) and TI ASC ('71), were memory-memory machines



Vector Memory-Memory vs. Vector Register Machines

- Vector memory-memory architectures (VMMA) require greater main memory bandwidth. Why?
- VMMA make it difficult to overlap execution of multiple vector operations. Why?
- VMMA incur greater startup latency.
 - Scalar code was faster on CDC Star-100 for vectors < 100 elements.
 - For Cray-1, vector/scalar breakeven point was around 2 elements.
- Apart from CDC follow-ons (Cyber-205, ETA-10) all major vector machines since Cray-1 have had *vector register architectures*.
 - We ignore vector memory-memory from now on.

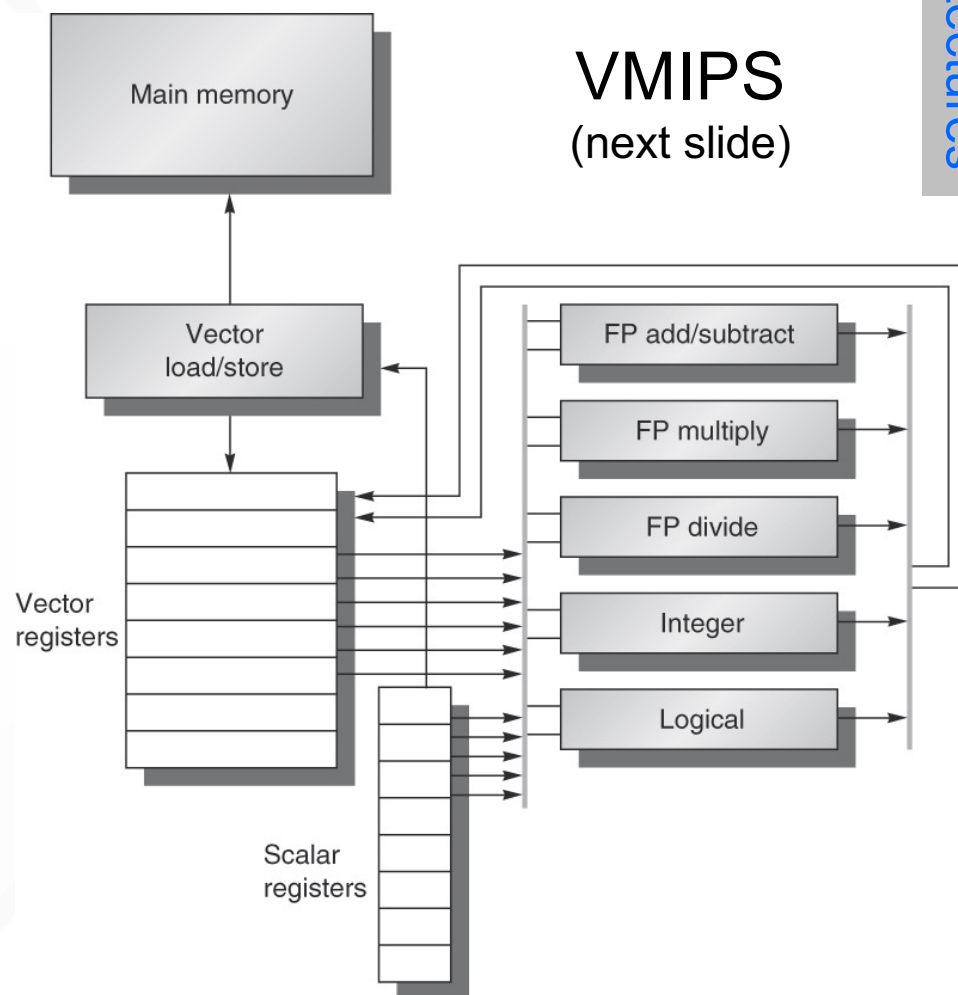
Vector Register Architectures

- Basic idea
 - Read sets of data elements into “vector registers”
 - Operate on those registers
 - Disperse the results back into memory
- Registers are controlled by compiler
 - Used to hide memory latency
 - Leverage memory bandwidth

Processor (year)	Vector clock rate (MHz)	Vector registers	Elements per register (64-bit elements)	Vector arithmetic units	Vector load-store units	Lanes
Cray-1 (1976)	80	8	64	6: FP add, FP multiply, FP reciprocal, integer add, logical, shift	1	1
Cray X-MP (1983)	118	8	64	8: FP add, FP multiply, FP reciprocal, integer add, 2 logical, shift, population count/parity	2 loads 1 store	1
Cray Y-MP (1988)	166	8	64	5: FP add, FP multiply, FP reciprocal/sqrt, integer add/shift/population count, logical	1	1
Fujitsu VP100/VP200 (1982)	133	8–256	32–1024	3: FP or integer add/logical, multiply, divide	2	1 (VP100) 2 (VP200)
Hitachi S810/S820 (1983)	71	32	256	4: FP multiply-add, FP multiply/divide-add unit, 2 integer add/logical	3 loads 1 store	1 (S810) 2 (S820)
Convex C-1 (1985)	10	8	128	2: FP or integer multiply/divide, add/logical	1	1 (64 bit) 2 (32 bit)
NEC SX/2 (1985)	167	8 + 32	256	4: FP multiply/divide, FP add, integer add/logical, shift	1	4
Cray C90 (1991)	240	8	128	8: FP add, FP multiply, FP reciprocal, integer add, 2 logical, shift, population count/parity	2 loads 1 store	2
Cray T90 (1995)	460	8 + 64	512	4: FP or integer add/shift, multiply, divide, logical	1	16
Fujitsu VPP5000 (1999)	300	8–256	128–4096	3: FP or integer multiply, add/logical, divide	1 load 1 store	16
Cray SV1 (1998)	300	8	64 (MSP)	8: FP add, FP multiply, FP reciprocal, integer add, 2 logical, shift, population count/parity	1 load-store 1 load	2 8 (MSP)
SV1ex (2001)	500	8	64	5: FP multiply, FP divide, FP add, integer add/shift, logical	1 load-store	1
VMIPS (2001)	500	8 + 64	256	4: FP or integer add/shift, multiply, divide, logical	1	8
NEC SX/6 (2001)	500	8 + 64	256	4: FP or integer add/shift, multiply, divide, logical	1	4
NEC SX/8 (2004)	2000	8 + 64	256	3: FP or integer, add/logical, multiply/shift, divide/square root/logical	1 load 1 store	2 8 (MSP)
Cray X1 (2002)	800	32	64	3: FP or integer, add/logical, multiply/shift, divide/square root/logical	1 load 1 store	2 8 (MSP)
Cray XIE (2005)	1130					

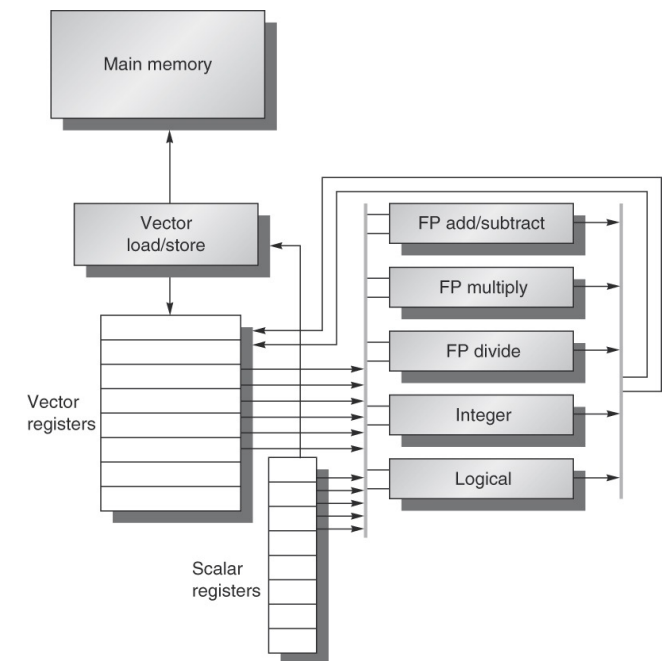
Vector Register Architectures

- Basic idea
 - Read sets of data elements into “vector registers”
 - Operate on those registers
 - Disperse the results back into memory
- Registers are controlled by compiler
 - Used to hide memory latency
 - Leverage memory bandwidth



VMIPS

- Example architecture: VMIPS
 - Loosely based on Cray-1
 - Vector registers
 - Each register holds a 64-element, 64 bits/element vector
 - Register file has 16 read ports and 8 write ports
 - Vector functional units
 - Fully pipelined
 - Data and control hazards are detected
 - Vector load-store unit
 - Fully pipelined
 - One word per clock cycle after initial latency
 - Scalar registers
 - 32 general-purpose registers
 - 32 floating-point registers



VMIPS Instructions

- ADDVV.D: add two vectors
- ADDVS.D: add vector to a scalar
- LV/SV: vector load and vector store from address
- Example: DAXPY

```
L.D      F0,a      ; load scalar a
LV       V1,Rx     ; load vector X
MULVS.D  V2,V1,F0  ; vector-scalar multiply
LV       V3,Ry     ; load vector Y
ADDVV    V4,V2,V3  ; add
SV       Ry,V4     ; store the result
```
- Requires 6 instructions vs. almost 600 for MIPS

Vector Execution Time

- Execution time depends on three factors:
 - Length of operand vectors
 - Structural hazards
 - Data dependencies
- 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