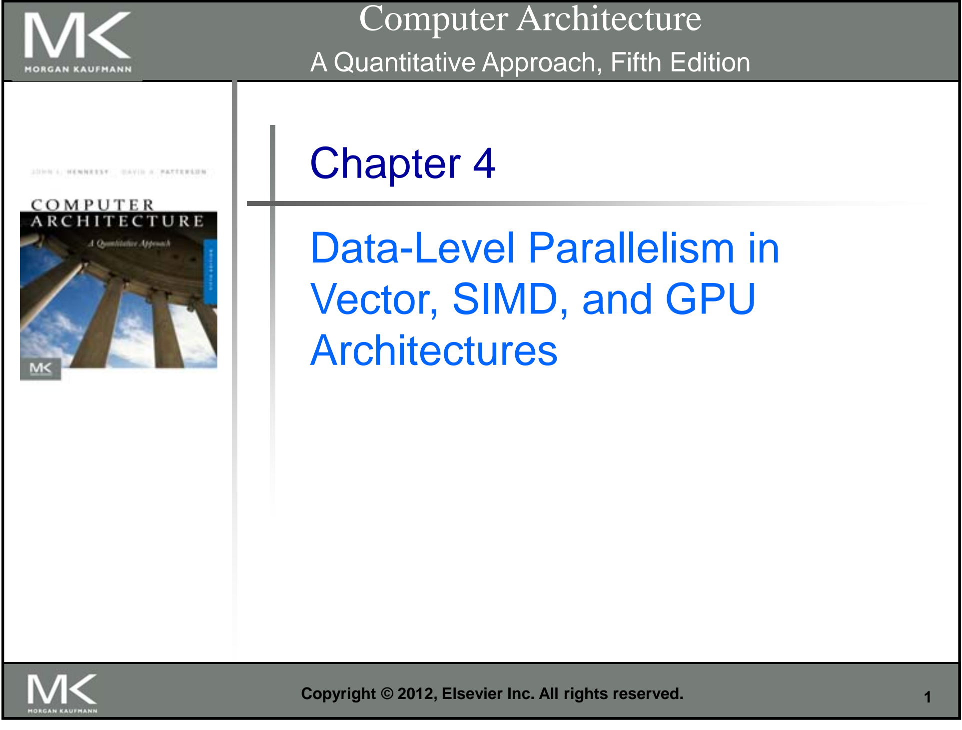


Introduction

- 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

Copyright © 2012, Elsevier Inc. All rights reserved.

SIMD Parallelism

Introduction

- Vector architectures
- SIMD extensions
- Graphics Processor Units (GPUs)

- For x86 processors:
  - Expect two additional cores per chip per year
  - SIMD width to double every four years
  - Potential speedup from SIMD to be twice that from MIMD!

Copyright © 2012, Elsevier Inc. All rights reserved.

Vector Architectures

Vector 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

Copyright © 2012, Elsevier Inc. All rights reserved.

VMIPS

Vector Architectures

- 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

Copyright © 2012, Elsevier Inc. All rights reserved. 5

VMIPS

Main memory
Vector load/store
Vector registers
Scalar registers
FP add/sub.
FP multiply
FP divide
Integer
Boolean

Copyright © 2012, Elsevier Inc. All rights reserved. 6

VMIPS Instructions

Vector Architectures

- ADDVV.D V1,V2,V3 add two vectors
- ADDVS.D V1,V2,F0 add vector to a scalar
- LV V1,R1 vector load from address
- SV R1,V1 Vector store at R1
- MULVV.D V1,V2,V3 vector multiply
- DIVVV.D V1,V2,V3 Vector div (element by element)
- LVWS V1,(R1,R2) Load vector from R1, stride=R2
- LVI V1,(R1+V2) Load V1 with elements at R1+V2(i)
- CVI V1,R1 load in V1 0,R1,2R1,3R1,…(index vector)
- SEQVV.D V1,V2 Compare elements V1,V2 0 or 1in VM
- MVTM VM,F0 Move contents of F0 to vec. mask reg.
- MTCI VLR,R1 Move r1 to vector length register

MK

Copyright © 2012, Elsevier Inc. All rights reserved.

VMIPS Instructions

- 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 (instruction bandwidth).
- Also, in MIPS must wait after LD and MUL (unless we do loop unrolling to avoid stalls).
- In vector architecture, we use chaining (what is the difference between chaining and forwarding?)

MK

Copyright © 2012, Elsevier Inc. All rights reserved.

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
- *Convey*
  - Set of vector instructions that could potentially execute together (could be more than one instruction)

Copyright © 2012, Elsevier Inc. All rights reserved.

Chimes

Vector Architectures

- Sequences with read-after-write dependency hazards can be in the same convey via *chaining*
- *Chaining*
  - Allows a vector operation to start as soon as the individual elements of its vector source operand become available
- *Chime*
  - Unit of time to execute one convey
  - *m* conveys executes in *m* chimes
  - For vector length of *n*, requires *m* x *n* clock cycles

Copyright © 2012, Elsevier Inc. All rights reserved.

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

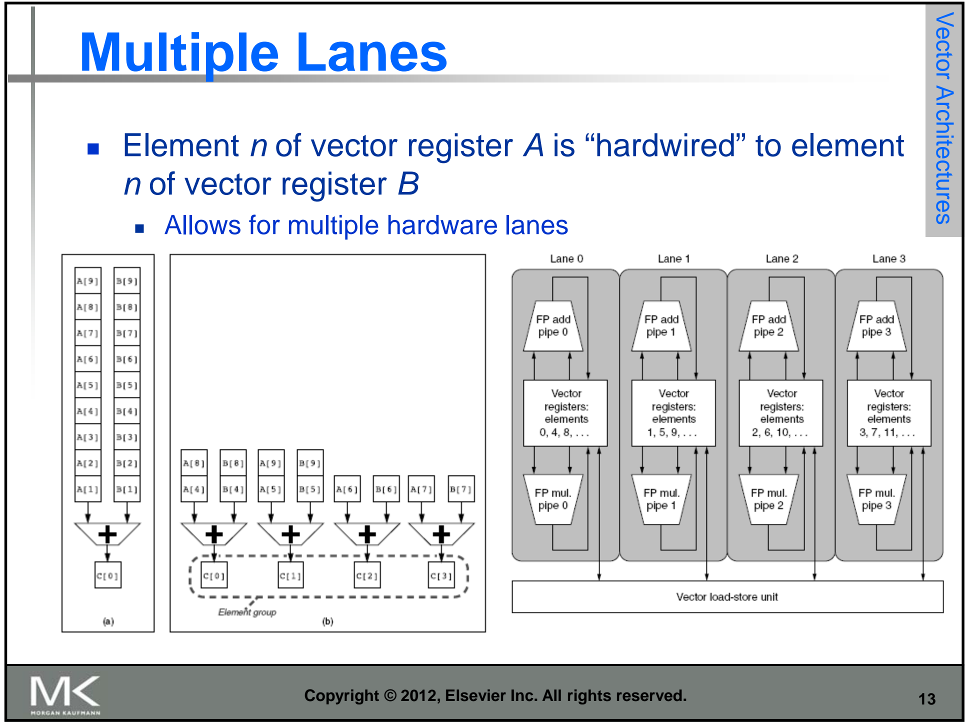
For 64 element vectors, requires 64 x 3 = 192 clock cycles

MK MORGAN KAUFMANN
Copyright © 2012, Elsevier Inc. All rights reserved.

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
- Improvements:
  - > 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

MK MORGAN KAUFMANN
Copyright © 2012, Elsevier Inc. All rights reserved.



Vector Length Register

- Vector length not known at compile time?
- Use Vector Length Register (VLR)
- 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*/
}
```

Value of j	0	1	2	3	. . .	. . .	n/MVL
Range of i	0 .. (m−1)	m .. (m−1) +MVL	(m+MVL) .. (m−1) +2×MVL	(m+2×MVL) .. (m−1) +3×MVL	. . .	. . .	(n−MVL) .. (n−1)

Vector Architectures

Copyright © 2012, Elsevier Inc. All rights reserved.

Vector Mask Registers

- What if we have a conditional IF statement inside the loop?
- Using scalar architecture, that introduces control dependence.
- The *vector-mask control*: A mask register is used to conditionally execute using a Boolean condition.
- When the *vector-mask register* is enabled, any vector instruction executed operate only on vector elements whose corresponding entries in the VMR are ones.
- The rest of the elements are unaffected.
- Clearing the vector mask register, sets to all 1's and operations are performed on all the elements.
- Does not save execution time for masked elements

Copyright © 2012, Elsevier Inc. All rights reserved.

Vector Mask Registers

- Consider:

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

- 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!

Copyright © 2012, Elsevier Inc. All rights reserved.

Memory Banks

- Load/store unit is more complicated than FU's
- Start-up time, is the time for the first word into a register
- 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?

MK MORGAN KAUFMANN
Copyright © 2012, Elsevier Inc. All rights reserved.

Stride

Vector Architectures

- Consider:

```
for (i = 0; i < 100; i=i+1)
    for (j = 0; j < 100; j=j+1) {
            A[i][j] = 0.0;
            for (k = 0; k < 100; k=k+1)
            A[i][j] = A[i][j] + B[i][k] * D[k][j];
    }
```

- Must vectorize multiplication of rows of B with columns of D
- Use *non-unit stride*
- Bank conflict (stall) occurs when the same bank is hit faster than bank busy time:
  - #banks / LCM(stride,#banks) < bank busy time

MK MORGAN KAUFMANN
Copyright © 2012, Elsevier Inc. All rights reserved.

Strides

Add in a bank					SE	Q		M	O	D
	0	1	2	3	0	1	2	0	1	2
0	0	1	2	3	0	1	2	0	16	8
1	4	5	6	7	3	4	5	9	1	17
2	8	9	10	11	6	7	8	18	10	2
3	12	13	14	15	9	10	11	3	19	11
4	16	17	18	19	12	13	14	12	4	20
5	20	21	22	23	15	16	17	21	13	5
6	24	25	26	27	18	19	20	6	22	14
7	28	29	30	31	21	22	23	15	7	23

MK MORGAN KAUFMANN
Copyright © 2012, Elsevier Inc. All rights reserved.

Strides

- MOD can be calculated very efficiently if the prime number is 1 less than a power of 2.
- Division still a problem
- But if we change the mapping such that
- Address in a bank = address MOD number of words in a bank.
- Since the number of words in a bank is usually a power of 2, that will lead to a very efficient implementation.
- Consider the following example, the first case is the usual 4 banks, then 3 banks with sequential interleaving and modulo interleaving and notice the conflict free access to rows and columns of a 4 by 4 matrix

MK MORGAN KAUFMANN
Copyright © 2012, Elsevier Inc. All rights reserved.

Scatter-Gather

- Consider:

```
for (i = 0; i < n; i=i+1)
    A[K[i]] = A[K[i]] + C[M[i]];
```

- Use index vector:

```
LV       Vk, Rk          ;load K
LVI      Va, (Ra+Vk)     ;load A[K[]]
LV       Vm, Rm          ;load M
LVI      Vc, (Rc+Vm)     ;load C[M[]]
ADDVV.D  Va, Va, Vc      ;add them
SVI      (Ra+Vk), Va     ;store A[K[]]
```

Vector Architectures

MK MORGAN KAUFMANN Copyright © 2012, Elsevier Inc. All rights reserved. 21

Programming Vec. Architectures

- Compilers can provide feedback to programmers
- Programmers can provide hints to compiler

Benchmark name	Operations executed in vector mode, compiler-optimized	Operations executed in vector mode, with programmer aid	Speedup from hint optimization
BDNA	96.1%	97.2%	1.52
MG3D	95.1%	94.5%	1.00
FLO52	91.5%	88.7%	N/A
ARC3D	91.1%	92.0%	1.01
SPEC77	90.3%	90.4%	1.07
MDG	87.7%	94.2%	1.49
TRFD	69.8%	73.7%	1.67
DYFESM	68.8%	65.6%	N/A
ADM	42.9%	59.6%	3.60
OCEAN	42.8%	91.2%	3.92
TRACK	14.4%	54.6%	2.52
SPICE	11.5%	79.9%	4.06
QCD	4.2%	75.1%	2.15

Vector Architectures

MK MORGAN KAUFMANN Copyright © 2012, Elsevier Inc. All rights reserved. 22

SIMD Extensions

- Media applications operate on data types narrower than the native word size
  - Example: disconnect carry chains to “partition” adder
- Limitations, compared to vector instructions:
  - Number of data operands encoded into op code
  - No sophisticated addressing modes (strided, scatter-gather)
  - No mask registers

Copyright © 2012, Elsevier Inc. All rights reserved.

SIMD Implementations

SIMD Instruction Set Extensions for Multimedia

- Implementations:
  - Intel MMX (1996)
    - Eight 8-bit integer ops or four 16-bit integer ops
  - Streaming SIMD Extensions (SSE) (1999)
    - Eight 16-bit integer ops
    - Four 32-bit integer/fp ops or two 64-bit integer/fp ops
  - Advanced Vector Extensions (2010)
    - Four 64-bit integer/fp ops
  - Operands must be consecutive and aligned memory locations

Copyright © 2012, Elsevier Inc. All rights reserved.

SIMD Implementation

- Easier to implement than Vector machines
  - Little cost to add registers and instructions
  - Require little extra state compared to vector machines (context switching).
  - Does not require the high memory bandwidth the vector machines do.
  - Does not have to deal with issues like a page fault in the middle of accessing 64 memory access.

MK MORGAN KAUFMANN
Copyright © 2012, Elsevier Inc. All rights reserved.