



中国科学技术大学
University of Science and Technology of China

计算机体系结构

周学海

xhzhou@ustc.edu.cn

0551-63606864

中国科学技术大学



Review: 向量体系结构

- **向量处理机基本概念**
 - 基本思想: 两个向量的对应分量进行运算, 产生一个结果向量
- **向量处理机基本特征**
 - VSIW-一条指令包含多个操作
 - 单条向量指令内所包含的操作相互独立
 - 以已知模式访问存储器-多体交叉存储系统
 - 控制相关少
- **向量处理机基本结构**
 - 向量指令并行执行
 - 向量运算部件的执行方式-流水线方式
 - 向量部件结构-多“道”结构-多条运算流水线
- **向量处理机性能评估**
 - 向量指令流执行时间: Convey, Chimes, Start-up time
 - 其他指标: R_{∞} , $N_{1/2}$, N_V
- **向量处理机性能优化**
 - 链接技术
 - 条件执行
 - 稀疏矩阵



第6章 Data-Level Parallelism in Vector, SIMD, and GPU Architectures

6.1 向量处理机模型

数据级并行的研究动机

数据级并行的种类

向量体系结构

向量处理模型

基本特性及结构

性能评估及优化

6.2-1 向量处理机模型优化

6.2-2 面向多媒体应用的SIMD指令集扩展

6.3-1 GPU-I

6.3-2 GPU-II

GPU简介

GPU的编程模型

GPU的存储系统



6.2-1 向量处理机模型优化

- **存储器访问**
- **链接技术**
- **条件执行**
- **稀疏矩阵**



DAXPY ($Y = a \times X + Y$)

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 Stride

- 假设处理顺序相邻的元素在存储器中不顺序存储。例如

```
do 10 i = 1,100
```

```
  do 10 j = 1,100
```

```
    A(i,j) = 0.0
```

```
    do 10 k = 1,100
```

```
10      A(i,j) = A(i,j) + B(i,k) * C(k,j)
```

- B 或 C 的两次访问不会相邻 (相隔800 bytes)
- **stride**: 向量中相邻元素间的距离
=> **LVWS** (load vector with stride) instruction
- Strides => 会导致体冲突
(e.g., stride = 32 and 16 banks)



Memory operations

- Load/store 操作成组地在寄存器和存储器之间移动数据
- 三类寻址方式

- Unit stride (单步长)

- Fastest

$LV\ V1, R1$ // $V1=M[R1..R1+63]$ load, stride=1

- Non-unit (constant) stride (常数步长)

$LVWS\ V1, R1, R2$ // $V1=M[R1..R1+63*R2]$ load, stride=R2

- Indexed (gather-scatter) (间接寻址)

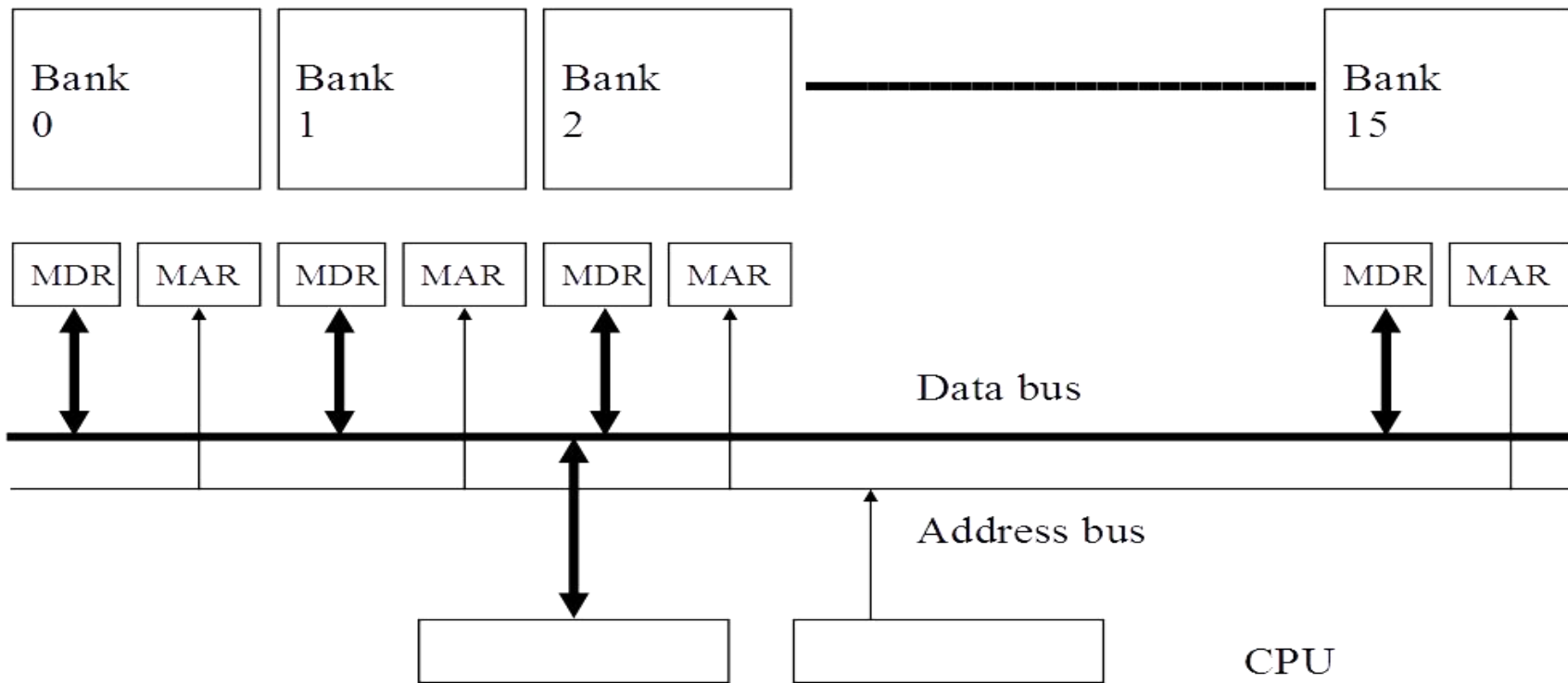
$LVI\ V1, R1, V2$ // $V1=M[R1+V2i, i=0..63]$ indir. ("gather")

- 等价于寄存器间接寻址方式
- 对稀疏矩阵有效
- 用于向量化操作的指令增多



Memory Banking

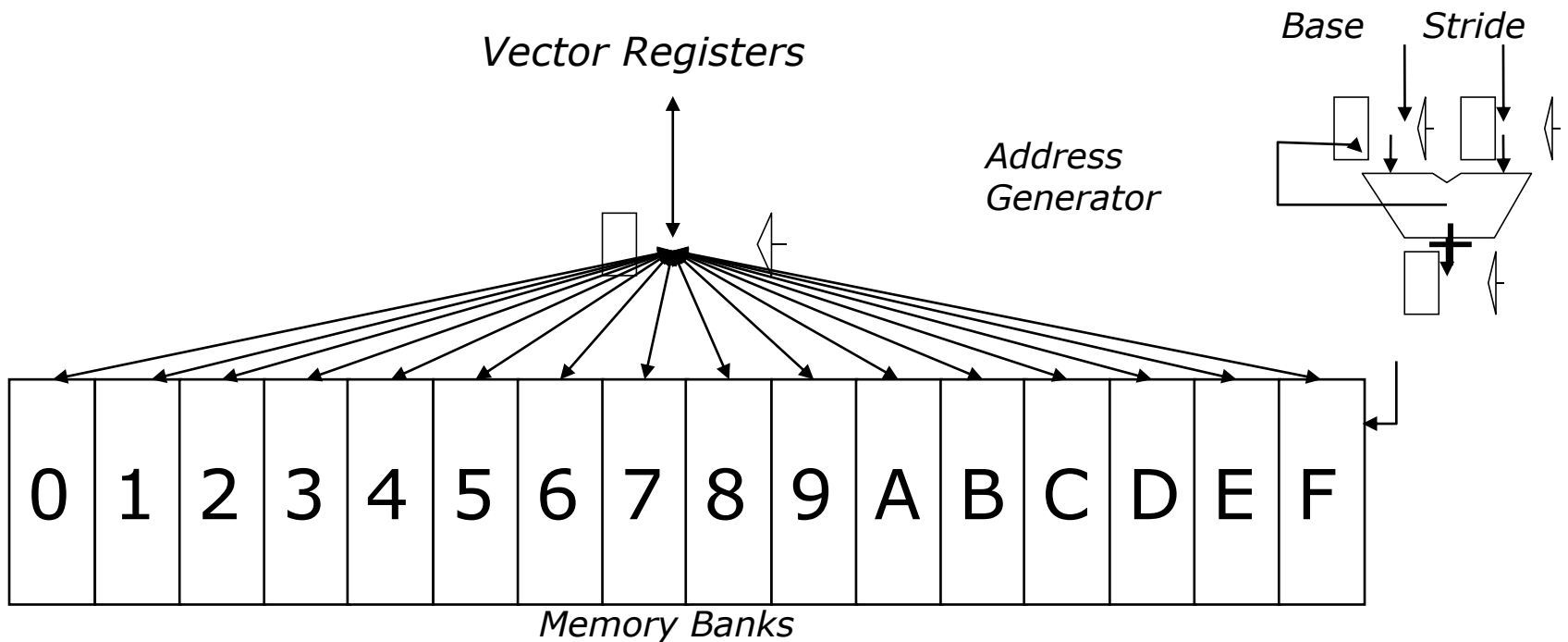
- **独立存储体方式：由多个相互独立的存储体 (Bank) 构成存储器组织。可独立访问存储体，各存储体共享数据和地址总线 (minimize pin cost)**
- **每个周期可以启动和完成一个bank的访问**
- **如果N个存储器访问不同的bank可以并行执行**





Interleaved Vector Memory System

- Cray-1, 16 banks, 4 cycle bank busy time, 12 cycle latency
 - *Bank busy time*: Time before bank ready to accept next request
 - *If stride = 1 & consecutive elements interleaved across banks & number of banks \geq bank latency, then can sustain 1 element/cycle throughput*





Example(AppF F-15) Suppose we want to fetch a vector of 64 elements starting at byte address 136, and a memory access takes 6 clocks. How many memory banks must we have to support one fetch per clock cycle? With what addresses are the banks accessed? When will the various elements arrive at the CPU?



Cycle no.	Bank							
	0	1	2	3	4	5	6	7
0		136						
1		busy	144					
2		busy	busy	152				
3		busy	busy	busy	160			
4		busy	busy	busy	busy	168		
5		busy	busy	busy	busy	busy	176	
6			busy	busy	busy	busy	busy	184
7	192			busy	busy	busy	busy	busy
8	busy	200			busy	busy	busy	busy
9	busy	busy	208			busy	busy	busy
10	busy	busy	busy	216			busy	busy
11	busy	busy	busy	busy	224			busy
12	busy	busy	busy	busy	busy	232		
13		busy	busy	busy	busy	busy	240	
14			busy	busy	busy	busy	busy	248
15	256			busy	busy	busy	busy	busy
16	busy	264			busy	busy	busy	busy

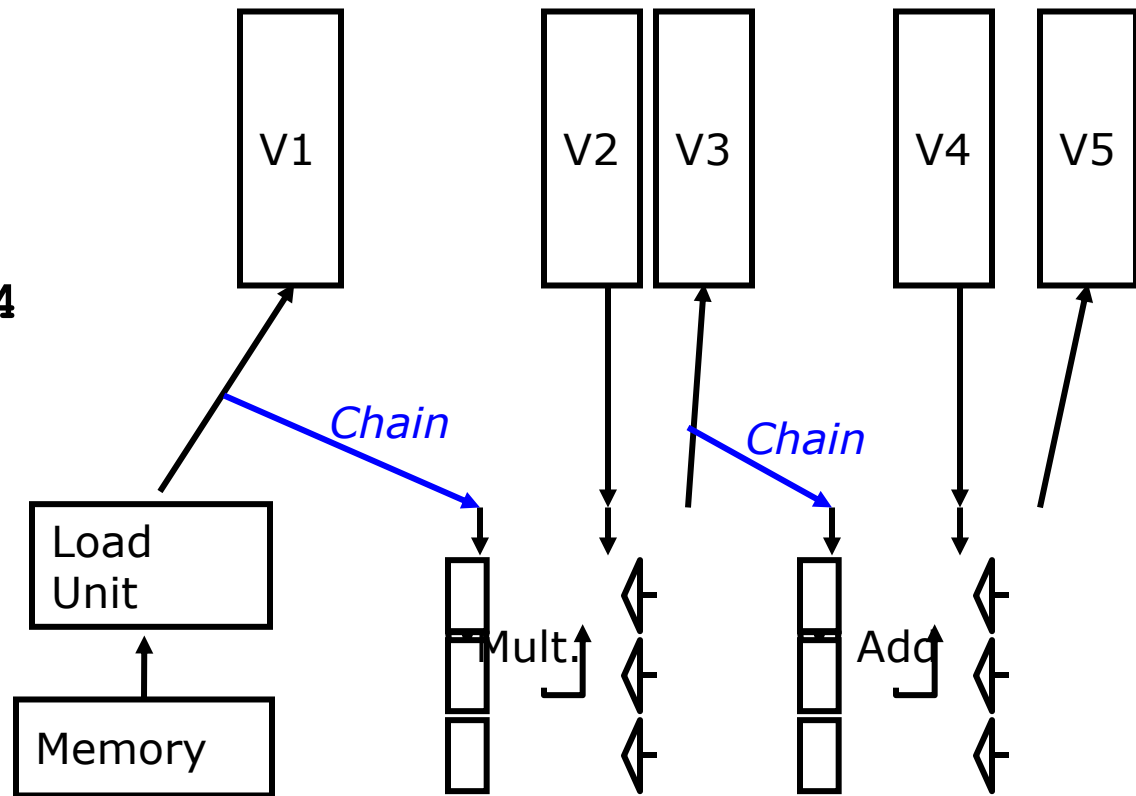
Figure F.7 Memory addresses (in bytes) by bank number and time slot at which access begins. Each memory bank latches the element address at the start of an access and is then busy for 6 clock cycles before returning a value to the CPU. Note that the CPU cannot keep all eight banks busy all the time because it is limited to supplying one new address and receiving one data item each cycle.



Vector Opt#1: Vector Chaining

- 寄存器定向路径的向量机版本
- 首次在Cray-1上使用

```
LV    v1  
MULV v3, v1, v2  
ADDV v5, v3, v4
```



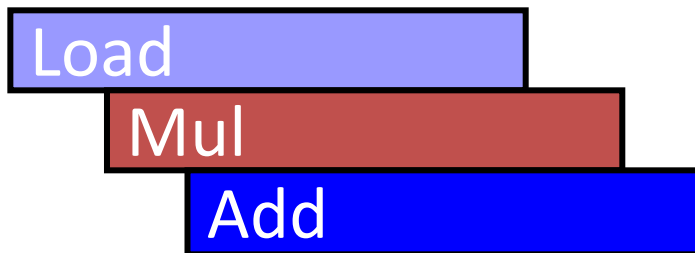


Vector Chaining Advantage

- 不采用链接技术，必须处理完前一条指令的最后一个元素，才能启动下一条相关的指令



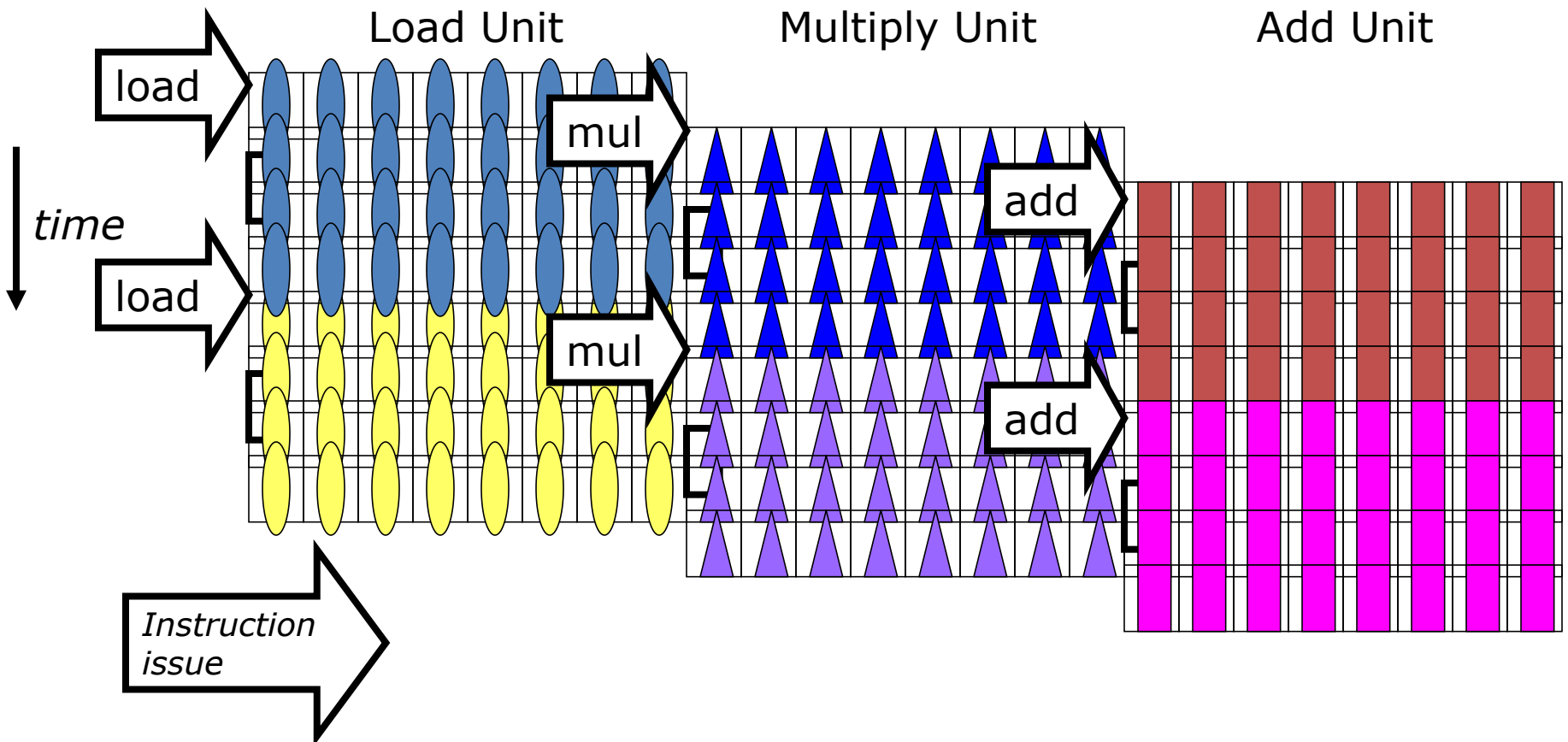
- 采用链接技术，前一条指令的第一个结果出来后，就可以启动下一条相关指令的执行





Vector Instruction Parallelism

- **多条向量指令可重叠执行(链接技术)**
 - 例如：每个向量 32 个元素，8 lanes (车道)



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



Vector Opt #2: Conditional Execution

- **Suppose:**

```
do 100 i = 1, 64
    if (A(i) .ne. 0) then
        A(i) = A(i) - B(i)
    endif
```

```
100 continue
```

- ***vector-mask control*** 使用长度为MVL的布尔向量控制向量指令的执行
- ***当vector-mask register*** 使能时，向量指令操作仅对vector-mask register中 对应位为1的分量起作用

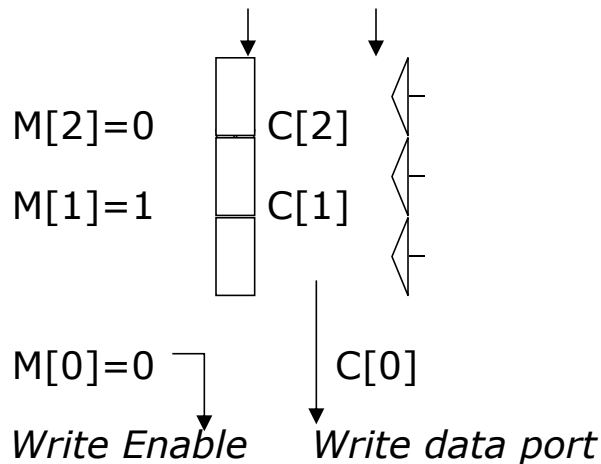


Masked Vector Instructions

Simple Implementation

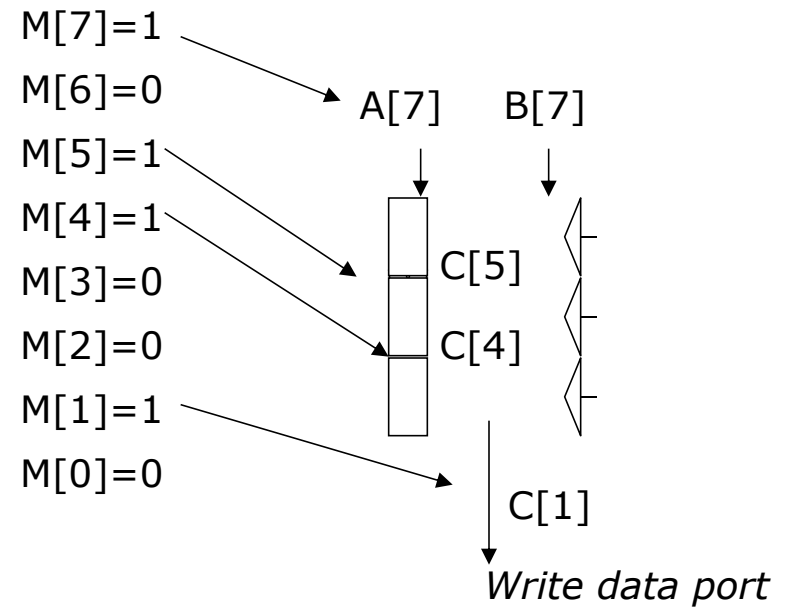
- execute all N operations, turn off result writeback according to mask

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]



Density-Time Implementation

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





LV V1,Ra	; load vector A into V1
LV V2,Rb	; load vector B
L.D F0,#0	; load FP zero into F0
SNEVS.D V1,F0	;sets VM(i) to 1 if V1(i)≠F0
SUBV.D V1,V1,V2	;subtract under vector mask
CVM	;set the vector mask to all 1s
SV Ra,V1	;store the result in A

- **使用vector-mask寄存器的缺陷**

- 简单实现时，条件不满足时向量指令仍然需要花费时间
- 有些向量处理器带条件的向量执行仅控制向目标寄存器的写操作，可能会有除法错。



Vector Opt #3: Sparse Matrices

- Suppose:

```
do 100 i = 1, n
100  A(K(i)) = A(K(i)) + C(M(i))
```

- ***gather* (LVI) operation** 使用 ***index vector*** 中给出的偏移再加基址来读取 => **a nonsparse vector in a vector register**
- 这些元素以密集的方式操作完成后，再使用同样的 **index vector** 存储到稀疏矩阵的对应位置
- 这些操作编译时可能无法完成。主要原因：编译器无法预知 $K(i)$ 以及是否有数据相关
- 使用 CVI 设置步长 (index 0, 1xm, 2xm, ..., 63xm)



Sparse Matrix Example

- **Cache (1993) vs. Vector (1988)**

	IBM RS6000	Cray YMP
Clock	72 MHz	167 MHz
Cache	256 KB	0.25 KB
Linpack	140 MFLOPS	160 (1.1)
Sparse Matrix (Cholesky Blocked)	17 MFLOPS	125 (7.3)

- **Cache: 1 address per cache block (32B to 64B)**
- **Vector: 1 address per element (4B)**



Summary: 向量体系结构

- **向量处理机基本概念**
 - 基本思想：两个向量的对应分量进行运算，产生一个结果向量
- **向量处理机基本特征**
 - VSIW-一条指令包含多个操作
 - 单条向量指令内所包含的操作相互独立
 - 以已知模式访问存储器-多体交叉存储系统
 - 控制相关少
- **向量处理机基本结构**
 - 向量指令并行执行
 - 向量运算部件的执行方式-流水线方式
 - 向量部件结构-多“道”结构-多条运算流水线
- **向量处理机性能评估**
 - **向量指令流执行时间**: Convey, Chimes, Start-up time
 - **其他指标**: R_{∞} , $N_{1/2}$, N_V
- **向量处理机性能优化**
 - 链接技术
 - 条件执行
 - 稀疏矩阵



Vector/SIMD Processing Summary

- **Vector/SIMD 机器适合挖掘数据级并行**
 - 同样的操作作用于不同的数据元素
 - 向量内的元素操作独立，可有效提高性能，简化设计
- **性能的提升受限于代码的向量化**
 - 标量操作限制了向量机的性能
 - Amdahl's Law
- **很多ISA包含SIMD操作指令**
 - Intel MMX/SSEn/AVX, PowerPC AltiVec, ARM Advanced SIMD



Array vs. Vector Processors

Array processor: 又称为并行处理机、SIMD处理器。其核心是一个由多个处理单元构成的阵列，用单一的控制部件来控制多个处理单元对各自的数据进行相同的运算和操作。

ARRAY PROCESSOR



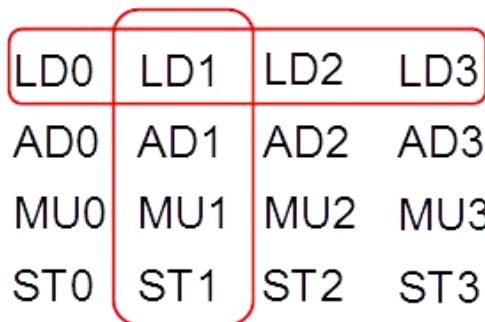
VECTOR PROCESSOR



Instruction Stream

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

Same op @ same time

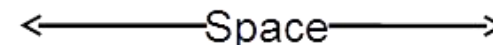
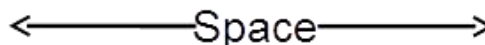


Different ops @ same space

Different ops @ time



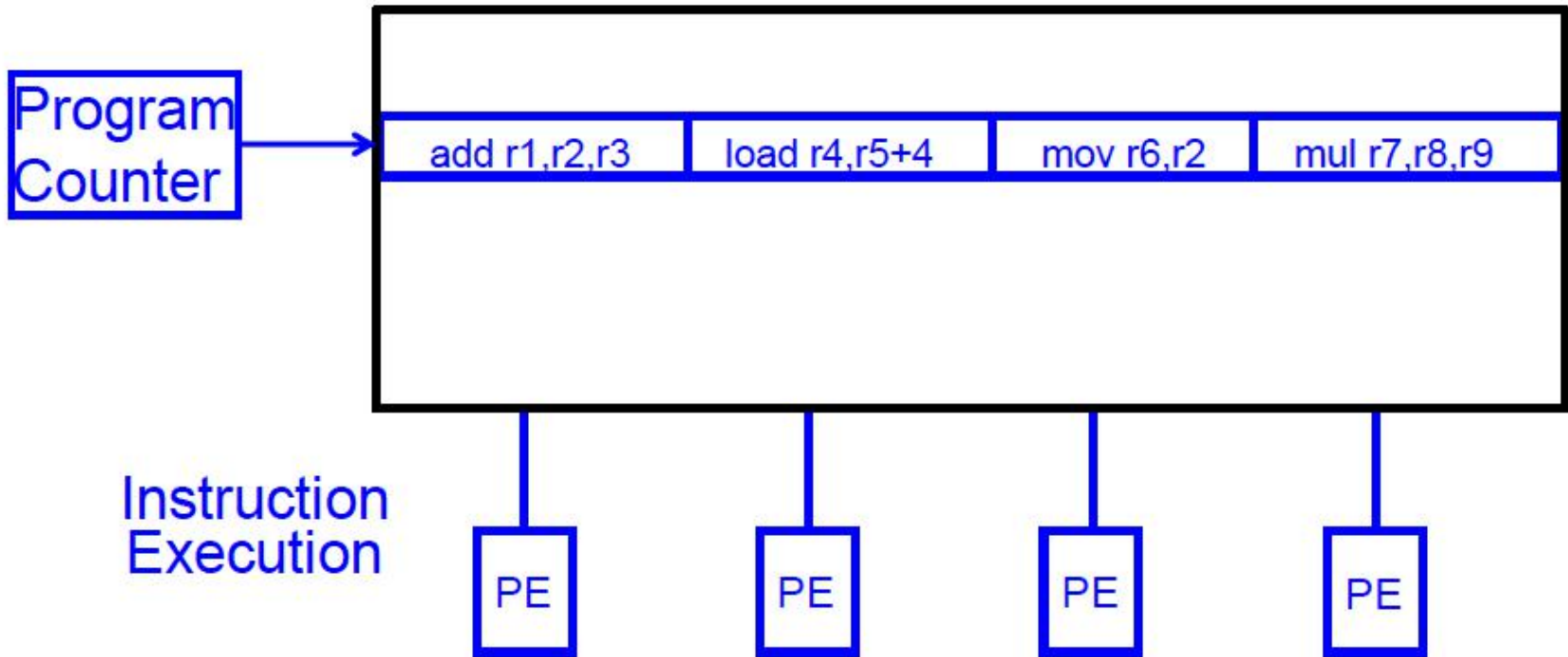
Same op @ space





SIMD Array Processing vs. VLIW

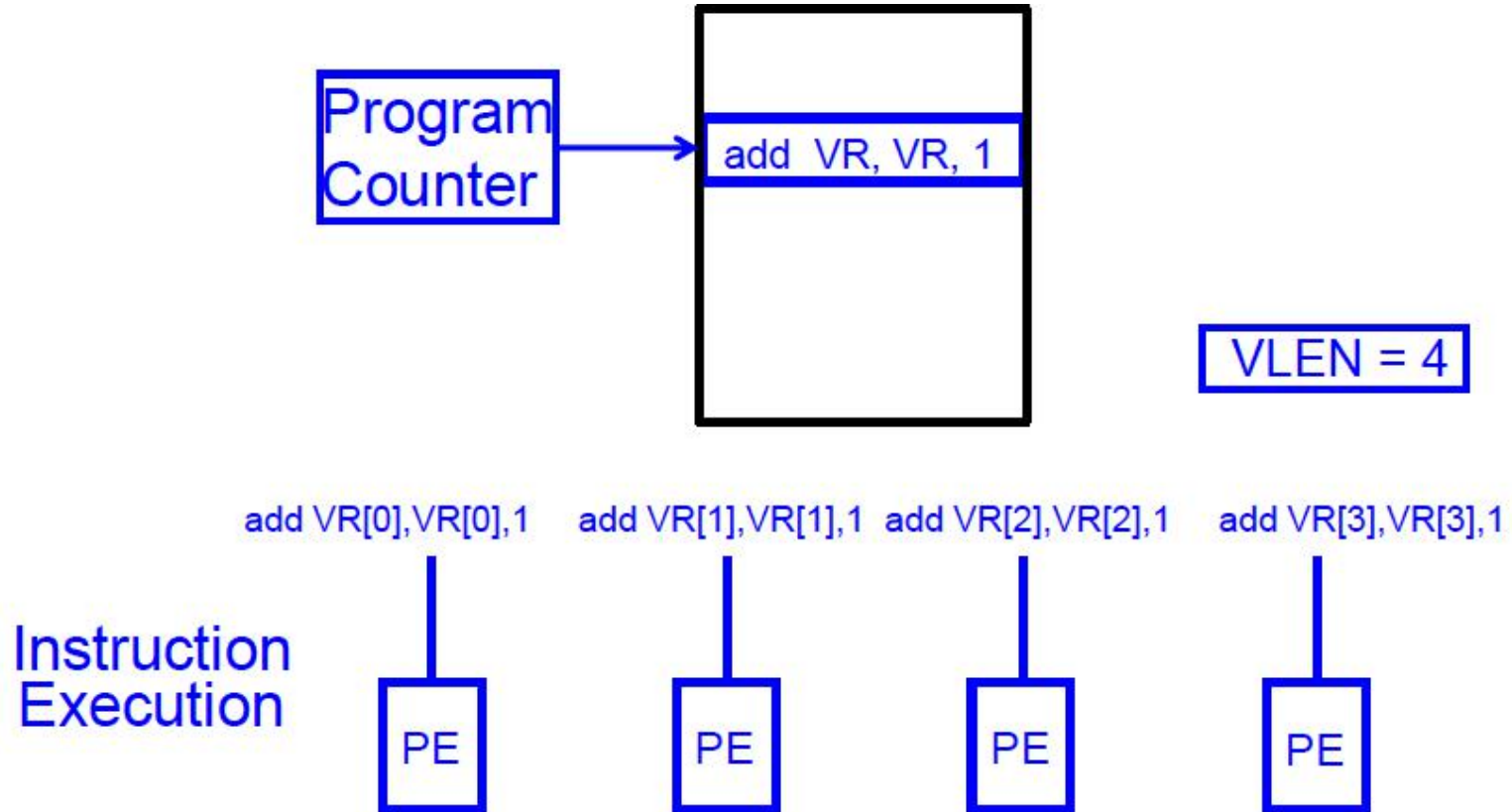
- VLIW: 多个独立的操作由编译器封装在一起





SIMD Array Processing vs. VLIW

- **Array processor: 单个操作作用在多个不同的数据元素上**





6.2-2 SIMD扩展

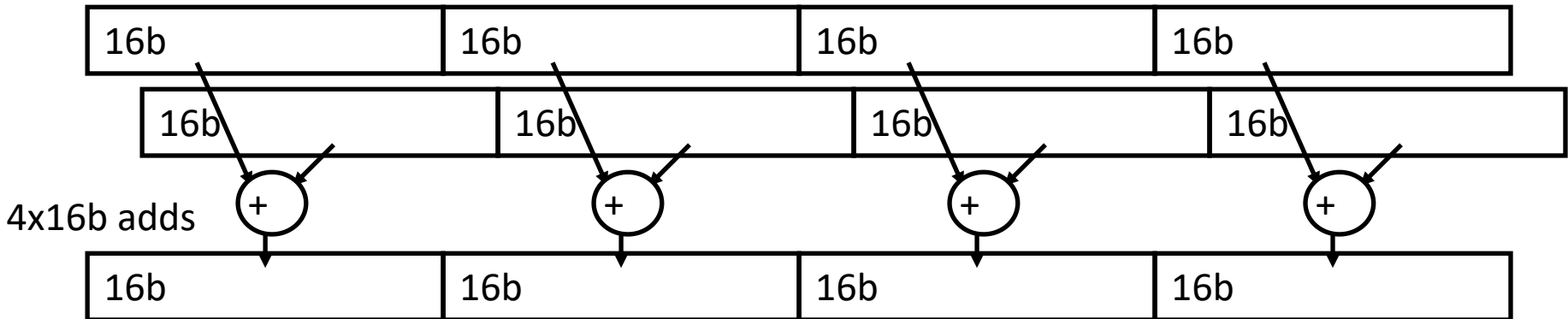
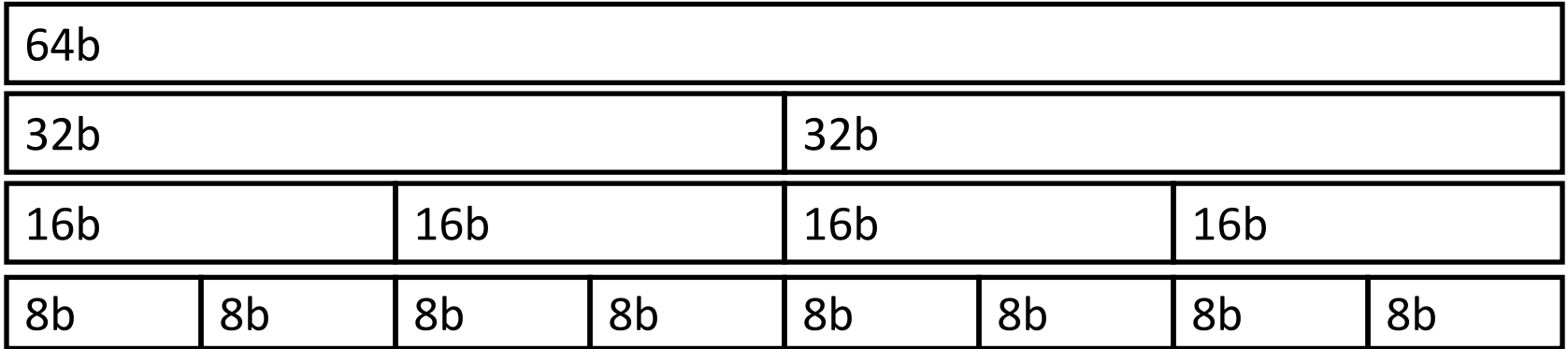


Multimedia Extensions (aka SIMD extensions)

- **在已有ISA中添加一些向量长度很短的向量操作指令**
- **将已有的 64-bit 寄存器拆分为 2x32b or 4x16b or 8x8b**
 - 1957年, Lincoln Labs TX-2 将36bit datapath 拆分为2x18b or 4x9b
 - 新的设计具有较宽的寄存器
 - 128b for PowerPC AltiVec, Intel SSE2/3/4
 - 256b for Intel AVX (Advanced Vector Extensions)
- **单条指令可实现寄存器中所有向量元素的操作**



Multimedia Extensions (aka SIMD extensions)





Intel Pentium MMX Operations

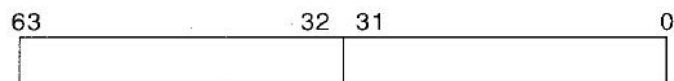
- **idea: 一条指令操作同时作用于不同的数据元**
 - 全阵列处理
 - 用于多媒体操作



(a)



(b)



(c)



(d)

- No VLEN register
- Opcode determines data type:
 - 8 8-bit bytes
 - 4 16-bit words
 - 2 32-bit doublewords
 - 1 64-bit quadword
- Stride always equal to 1.

Figure 1. MMX technology data types: packed byte (a), packed word (b), packed doubleword (c), and quadword (d).

MMX Example: Image Overlaying (I)



Figure 8. Chroma keying: image overlay using a background color.

PCMPEQB MM1, MM3

MM1	Blue	Blue	Blue	Blue	Blue	Blue	Blue	Blue
MM3	X7!=blue	X6!=blue	X5=blue	X4=blue	X3!=blue	X2!=blue	X1=blue	X0=blue
MM1	0x0000	0x0000	0xFFFF	0xFFFF	0x0000	0x0000	0xFFFF	0xFFFF



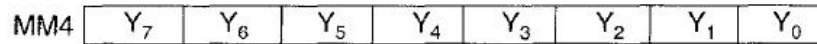
Bitmask

Figure 9. Generating the selection bit mask.



MMX Example: Image Overlaying (II)

PAND MM4, MM1



PANDN MM1, MM3

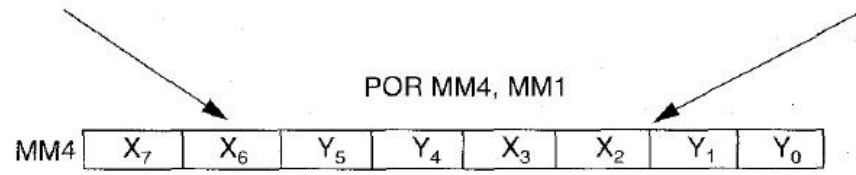
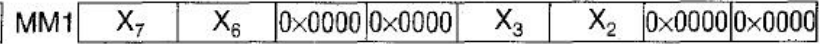
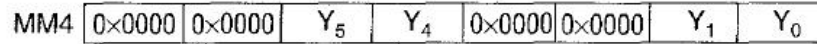
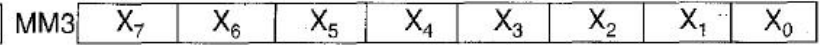


Figure 10. Using the mask with logical MMX instructions to perform a conditional select.

```

Movq    mm3, mem1    /* Load eight pixels from
                    woman's image
Movq    mm4, mem2    /* Load eight pixels from the
                    blossom image
Pcmpeqb mm1, mm3
Pand    mm4, mm1
Pandn   mm1, mm3
Por     mm4, mm1
    
```

Figure 11. MMX code sequence for performing a conditional select.



Multimedia Extensions versus Vectors

- **受限的指令集:**
 - 无向量长度控制
 - Load/store操作无 常数步长寻址和 scatter/gather操作
 - loads 操作必须64/128-bit 边界对齐
- **受限的向量寄存器长度:**
 - 需要超标量发射以保持multiply/add/load 部件忙
 - 通过循环展开隐藏延迟增加了寄存器读写压力
- **在微处理器设计中向全向量化发展**
 - 更好地支持非对齐存储器访问
 - 支持双精度浮点数操作 (64-bit floating-point)
 - Intel AVX spec (announced April 2008), 256b vector registers (expandable up to 1024b)



Summary

- **向量机的存储器访问**
 - 存储器组织：独立存储体、多体交叉方式
 - Stride：固定步长（1 or 常数），非固定步长（index）
- **基于向量机模型的优化**
 - 链接技术
 - 有条件执行
 - 稀疏矩阵的操作
- **多媒体扩展指令**
 - 扩展的指令类型较少
 - 向量寄存器长度较短



并行的类型

- **指令级并行(ILP)**
 - 以并行方式执行某个指令流中的独立无关的指令 (pipelining, superscalar, VLIW)
- **数据级并行(DLP)**
 - 以并行方式执行多个相同类型的操作 (vector/SIMD execution)
 - Array Processor 、 Vector Processor
- **线程级并行 (TLP)**
 - 以并行方式执行多个独立的指令流 (multithreading, multiple cores)
- **Which is easiest to program?**
- **Which is most flexible form of parallelism?**
 - i.e., can be used in more situations
- **Which is most efficient?**
 - i.e., greatest tasks/second/area, lowest energy/task



随堂测验



Acknowledgements

- **These slides contain material developed and copyright by:**
 - John Kubiawicz (UCB)
 - Krste Asanovic (UCB)
 - John Hennessy (Stanford) and David Patterson (UCB)
 - Chenxi Zhang (Tongji)
 - Muhamed Mudawar (KFUPM)
- **UCB material derived from course CS152、CS252、CS61C**
- **KFUPM material derived from course COE501、COE502**