

第6章 向量处理机

肖梦白

xiaomb@sdu.edu.cn

<https://xiaomengbai.github.io>

- 6.1 向量数据表示方式
- 6.2 向量处理机的结构
- 6.3 向量处理方式
- 6.4 向量处理机的关键技术
- 6.5 向量处理机实例
- 6.6 向量处理机的性能评价
- 6.7 向量处理机的发展

- 向量处理机是具有向量数据表示和向量指令系统的处理机
- 向量处理机是解决数值计算问题的一种高性能计算机
- 向量处理机属大型或巨型机，也可以用微机加一台向量协处理器组成
- 向量处理机一般都采用流水线结构，通常有有多条并行工作的流水线
- 必须把要解决的问题转化为向量运算，才能发挥向量处理机的效率

6. 1 向量数据表示方式

6. 1. 1 从标量到向量

6. 1. 2 等间距向量表示法

6. 1. 3 带位移量的向量表示法

6. 1. 4 稀疏向量表示法

6.1.1 从标量到向量

一个简单的C语言程序如下：

```
for (i = 10; i <= 1010; i++)
```

```
    c[i] = a[i] + b[i];
```

- 在向量处理机上, 可以只用一条指令：

$C(10:1010) = A(10:1010) + B(10:1010)$

一条向量指令可处理N个或N对操作数

- 在标量处理机上用10多条指令，其中有8条指令要循环1000次。
- 采用多寄存器结构的两地址指令编写程序
- 存储器采用字节编址方式，字长为32位

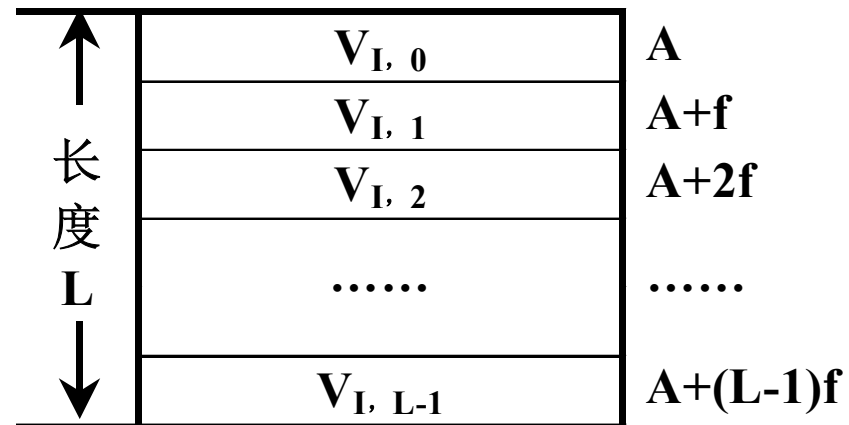
在一般标量处理机中需要如下指令序列来实现（A、B、C分别是向量a、b、c在内存中的起始地址）：

START:	LOAD	R0,	ST	;读循环初值10
	LOAD	R1,	ED	;读循环终值1010
	LOAD	R2,	L	;读内存地址增量4
	MOVE	R3,	R2	
	MUL	R3,	R0	;计算向量偏移量,
				;初始值为40
LOOP:	LOAD	R4,A(R3)		;读A向量的一个元素
	LOAD	R5, B(R3)		;读B向量的一个元素
	ADD	R4, R5		;加一个元素
	STORE	R4, C(R3)		;写C向量的一个元素
	ADD	R3, R2		;改变向量偏移量
	INC	R0		;循环次数增1
	CMP	R0, R1		;循环是否结束
	BLE	LOOP		;循环未结束转LOOP,
				;否则继续
	HALT			;停机
	ST: 10			;循环初值
	ED: 1010			;循环终值
	L: 4			;内存地址增量

6.1.2 等间距向量表示法

➤ 三个参数表示一个等间距向量：

- 向量起始地址：A
- 向量长度：L
- 向量间距：f



6.1.3 带位移量的向量表示法

➤ 用三个参数表示一个向量：

- 向量基地址：A
- 向量长度：L
- 向量间距：f
- 向量位移量：w
- 向量有效长度：L-w
- 向量起始地址：A+w



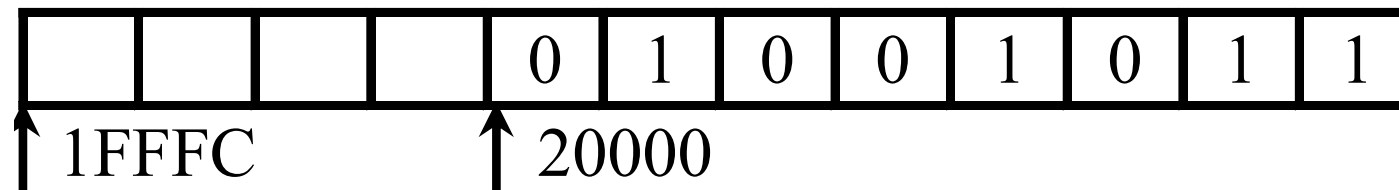
A 向量：A=10000，L=12，w=4

➤ 优点：

- 每个向量可以带有位移，能够通过控制向量实现可变增量。
- 能够表示稀疏向量。

1FF80	B-4	起始地址 ↑ 位移量 -4 ↓ 基地址 有效长度 8 ↓
1FFA0	B-3	
1FFC0	B-2	
1FFE0	B-1	
20000	B0	
20020	B1	
20040	B2	
20060	B3	

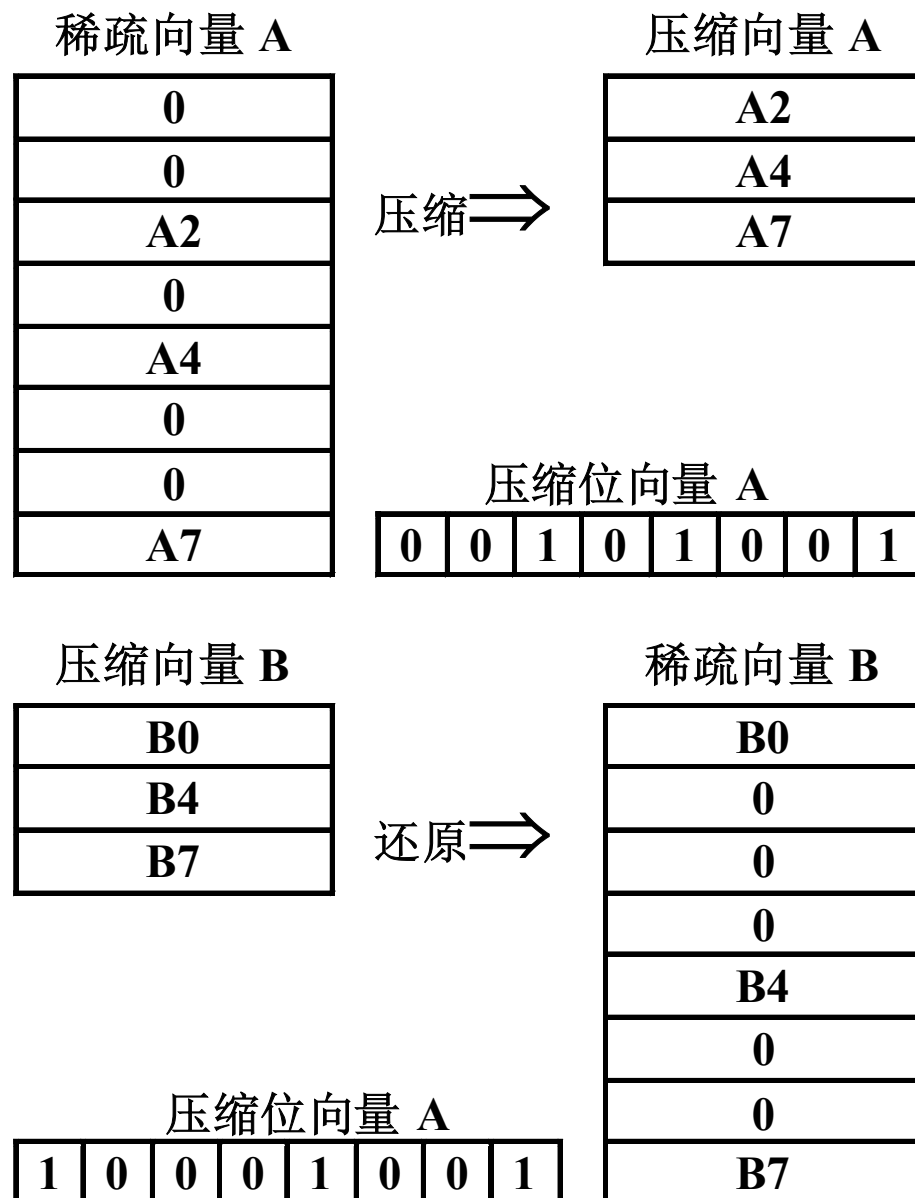
B 向量: $A=20000$, $L=4$, $f=32$, $w=-4$



控制向量: $A=20000$, $L=12$, $f=1$, $w=4$

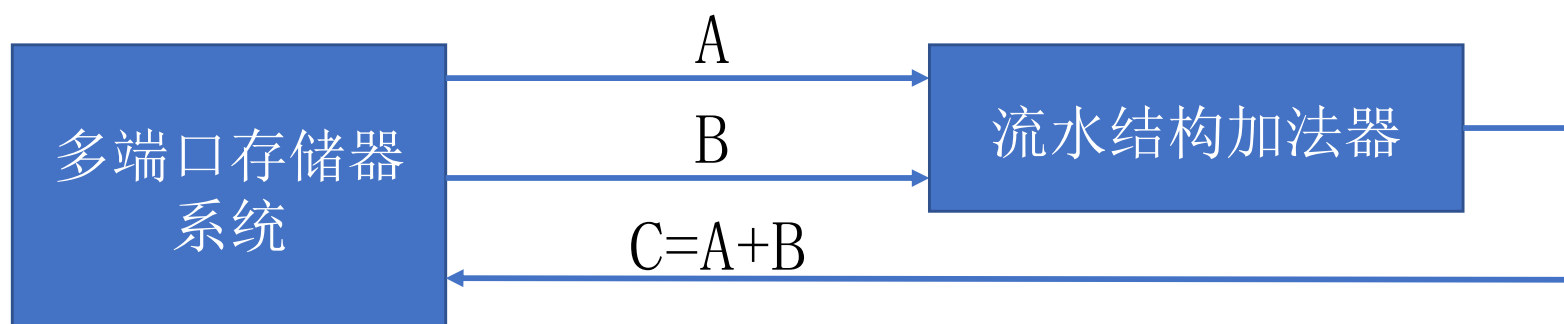
6.1.4 稀疏向量表示法

- 定义：0元素很多，非0元素很少的向量称为稀疏向量
- 采用压缩方法存储稀疏向量可以节省存储空间。
- 可以还原之后进行运算，也可以用压缩方法直接进行运算



6.2 向量处理机的结构

➤ 以N维向量加法为例 $C = A + B$



- 困难之处在于使存储器系统能够提供给运算器连续不断的数据流以及接受来自运算器的连续不断的运算结果
- 上图所示的存储系统的带宽至少3倍于一般存储器系统，因此另外一个主要问题是如何设计一个能满足运算器带宽要求的存储系统

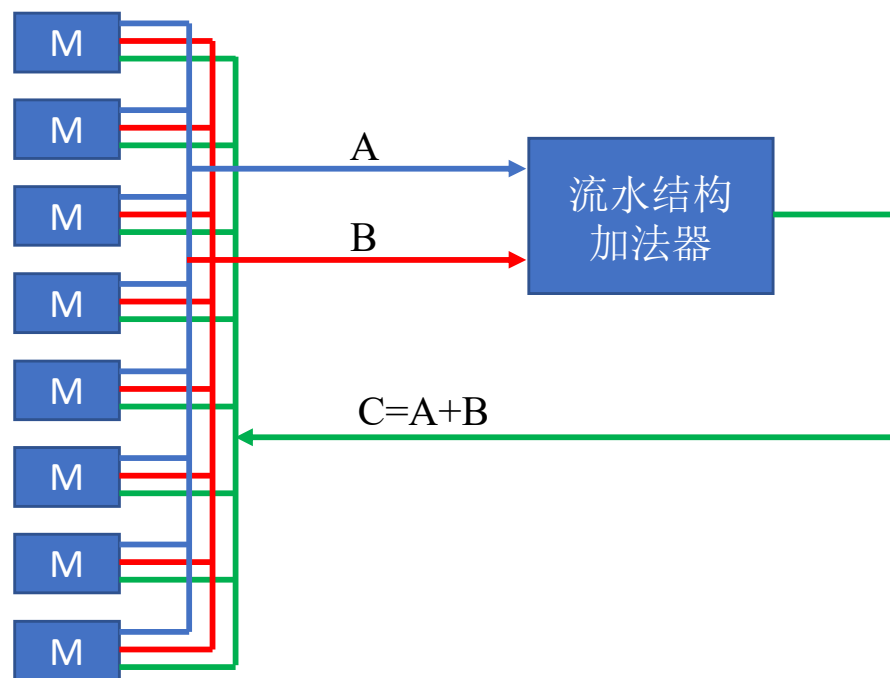
6.2 向量处理机的结构

- 存储器—存储器结构：利用几个独立的存储器模块来支持相对独立的数据并发访问，从而达到所要求的存储器带宽
 - 多个独立的存储器模块并行工作
 - 处理机结构简单
 - 对存储系统的访问速度要求很高
- 寄存器—寄存器结构：构造一个具有所要求带宽的高速中间存储器，并实现该高速中间存储器与主存储器之间的快速数据交换
 - 运算通过向量寄存器进行
 - 需要大量高速寄存器
 - 对存储系统访问速度的要求降低

6.2 向量处理机的结构

1. 存储器—存储器结构

- 向量处理机中有多个高速流水线运算部件，存储器的访问速度是关键
- 采用多个存储体交叉和并行访问来提高存储器速度
 - CRAY-1有64个存储体，每个处理机访问4个存储体
 - STAR-100采用32个存储体交叉，每个存储体并行读出8个64位数据
 - 我国研制的YH-1向量计算机有37个存储体



- 一个具有由8个三端口存储器组成的存储器系统的向量处理机
- 存储系统由8个模块构成，带宽是单个模块的8倍
- 流水线运算器与主存储器之间有三条相互独立的数据通路，各个数据通路可以同时工作，但一个存储器模块在某一时刻只能为一个通路服务
- 假设1个存储周期占2个处理机周期，则满足流水线所需的带宽至少是单个存储器模块的6倍

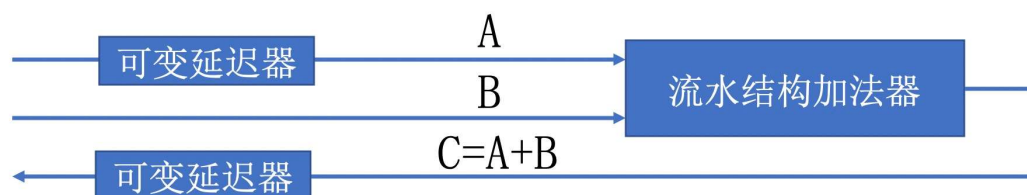
for (i = 0; i < 8; i++)

c[i] = a[i] + b[i] ;

模块0	A[0]		B[6]			C[4]
模块1	A[1]		B[7]			C[5]
模块2	A[2]	B[0]				C[6]
模块3	A[3]	B[1]				C[7]
模块4	A[4]	B[2]			C[0]	
模块5	A[5]	B[3]			C[1]	
模块6	A[6]	B[4]			C[2]	
模块7	A[7]	B[5]			C[3]	

流水线段4						0	1	2	3	4	5	6	7
流水线段3					0	1	2	3	4	5	6	7	
流水线段2				0	1	2	3	4	5	6	7		
流水线段1			0	1	2	3	4	5	6	7			
存储器7						RB5	RB5	RA7	RA7	W3	W3		
存储器6					RB4	RB4	RA6	RA6	W2	W2			
存储器5				RB3	RB3	RA5	RA5	W1	W1				
存储器4			RB2	RB2	RA4	RA4	W0	W0					
存储器3		RB1	RB1	RA3	RA3								
存储器2	RB0	RB0	RA2	RA2									W6
存储器1		RA1	RA1					RB7	RB7			W5	W5
存储器0	RA0	RA0					RB6	RB6			W4	W4	
	0	1	2	3	4	5	6	7	8	9	10	11	12
时间（时钟周期）													

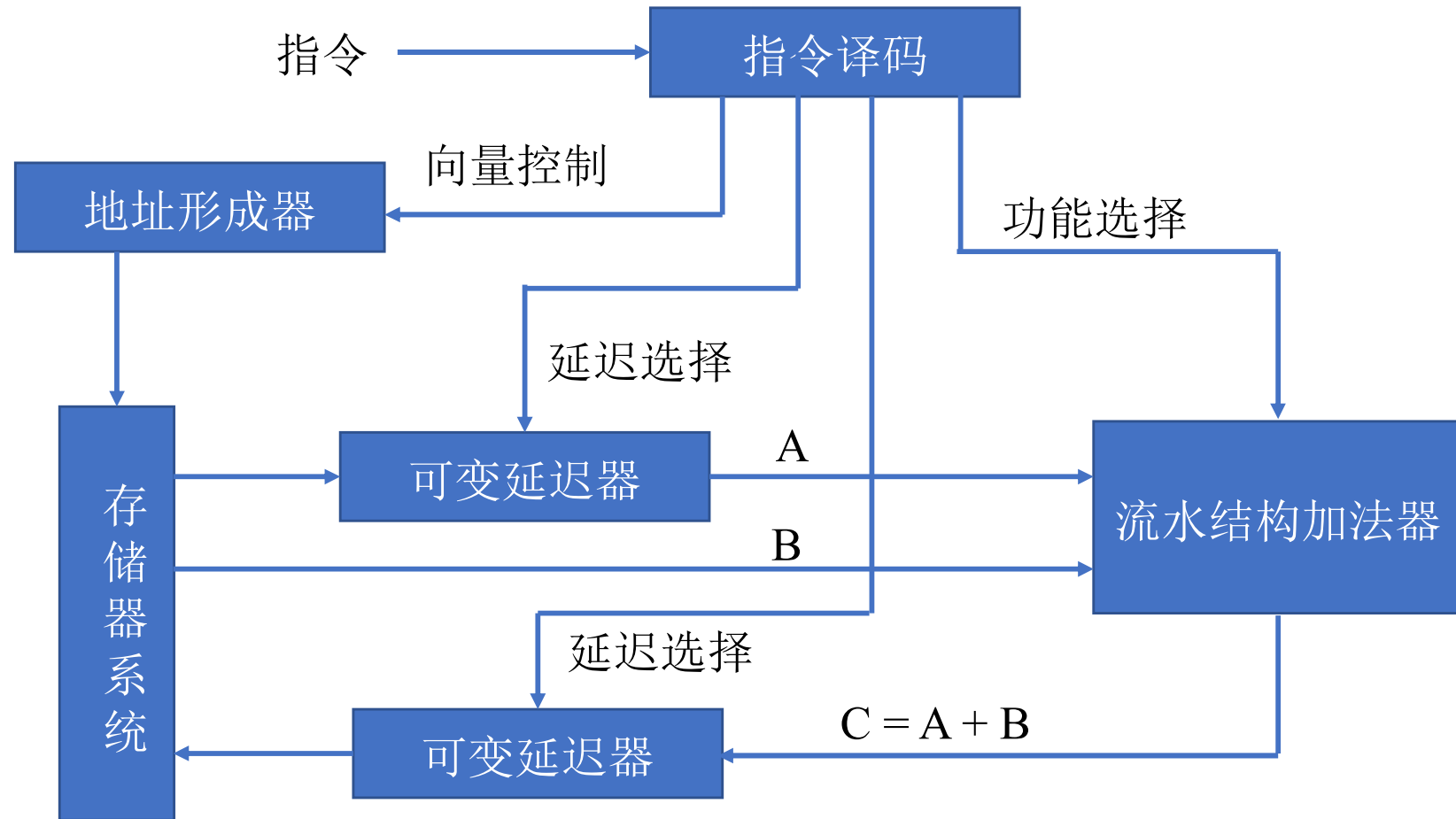
- 在运算流水线的输入端和输出端增加缓冲器以便消除争用存储器现象



- 假设所有向量都从模块0开始存放

流水线段4							0	1	2	3	4	5	
流水线段3						0	1	2	3	4	5	6	
流水线段2					0	1	2	3	4	5	6	7	
流水线段1				0	1	2	3	4	5	6	7		
存储器7							RA7	RA7	RA7	RA7			
存储器6						RA6	RA6	RB6	RB6				
存储器5					RA5	RA5	RB5	RB5					
存储器4				RA4	RA4	RB4	RB4					...	
存储器3			RA3	RA3	RB3	RB3					
存储器2		RA2	RA2	RB2	RB2					
存储器1		RA1	RA1	RB1	RB1				RA9	RA9	RB9	RB9	
存储器0	RA0	RA0	RB0	RB0					RA8	RA8	RB8	RB8	W0
	0	1	2	3	4	5	6	7	8	9	10	11	12
	时间（时钟周期）												

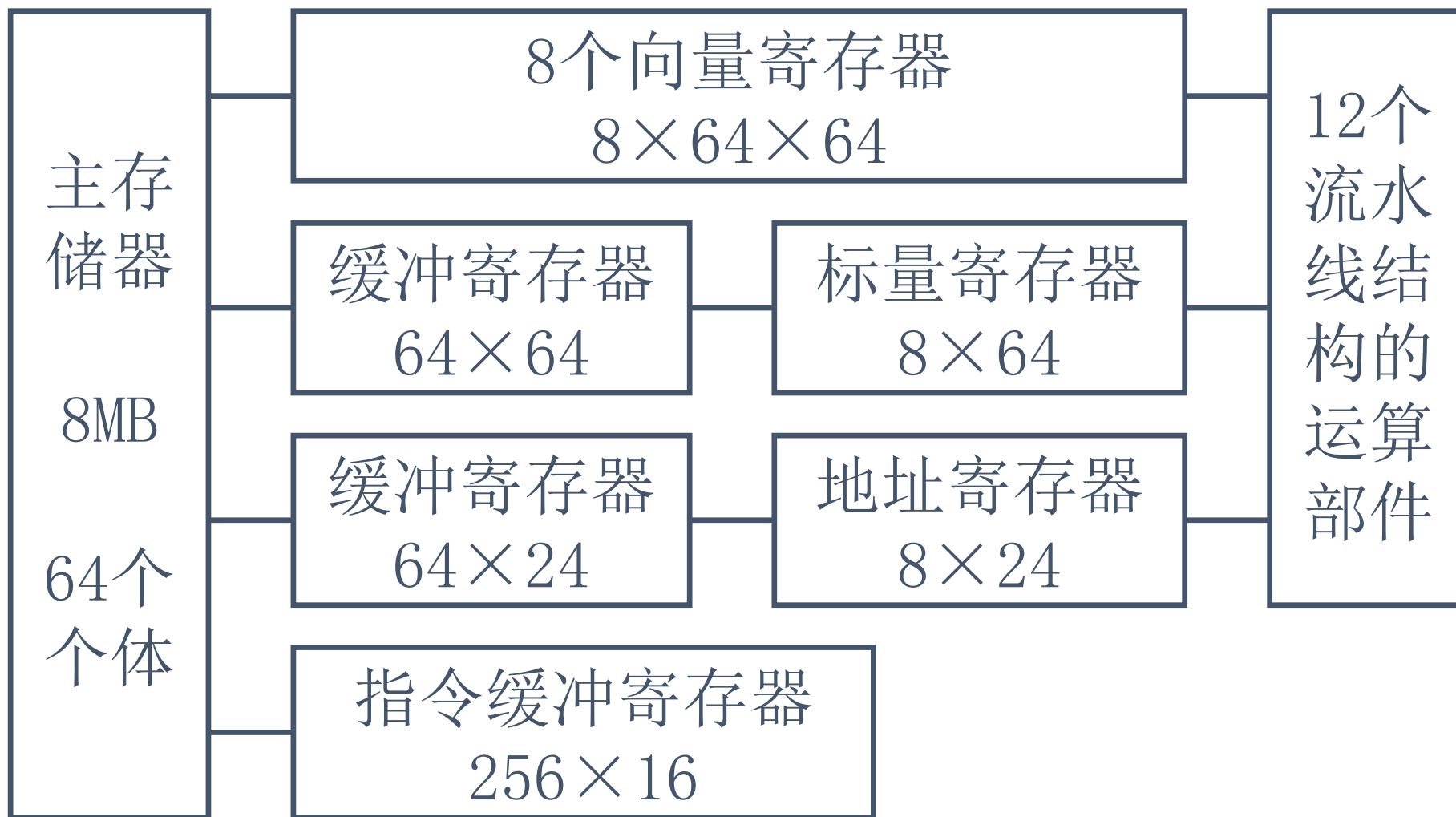
CDC STAR 巨型机



6.2 向量处理机的结构

2. 寄存器-寄存器结构

- 把存储器-存储器结构中的缓冲栈改为向量寄存器
- 运算部件需要的操作数从向量寄存器中读取，运算的中间结果也写到向量寄存器中。
- 向量寄存器与标量寄存器的主要差别是：一个向量寄存器能够保存一个向量，连续访问一个向量的各个分量。
- 需要有标量寄存器和地址寄存器等。



CRAY-1向量处理机结构

6.3 向量处理方式

- 横向处理方式，又称为水平处理方式，横向加工方式等。向量计算是按行的方式从左至右横向地进行。
- 纵向处理方式，又称为垂直处理方式，纵向加工方式等。向量计算是按列的方式自上而下纵向地进行。
- 纵横处理方式，又称为分组处理方式，纵横向加工方式等。横向处理和纵向处理相结合的方式。

- 要根据向量运算的特点和向量处理机的类型选择向量的处理方式。
- 以一个简单的C语言编写的程序为例，说明向量的三种处理方式的工作原理。

```
for (i = 1; i <= n; i++)  
    y[i] = a[i] × ( b[i] + c[i] );
```

6.3 向量处理方式-横向处理方式

➤ 逐个分量进行处理:

计算第1个分量: $T(1) = B(1) + C(1); Y(1) = A(1) \times T(1)$

计算第2个分量: $T(2) = B(2) + C(2); Y(2) = A(2) \times T(2)$

.....

计算最后一个分量: $T(N) = B(N) + C(N); Y(N) = A(N) \times T(N)$

➤ 存在的问题:

- 在计算向量的每个分量时, 都发生写读数据相关, 流水线效率低
- 如果采用多功能流水线, 必须频繁进行流水线切换

6.3 向量处理方式-纵向处理方式

➤ 也称为垂直处理方式，纵向加工方式等

$$T(1) = B(1) + C(1)$$

$$T(2) = B(2) + C(2)$$

.....

$$T(n) = B(n) + C(n)$$

$$Y(1) = A(1) \times T(1)$$

$$Y(2) = A(2) \times T(2)$$

.....

$$Y(N) = A(N) \times T(N)$$

6.3 向量处理方式-纵向处理方式

➤对整个向量按相同的运算处理完之后，再去执行别的运算，也称为垂直处理方式，纵向加工方式等

➤采用向量指令只需要2条：

VADD B, C, T

VMUL A, T, Y

➤每条向量指令内都是单一相同的运算，两条向量指令间仅有1次功能切换，适用于向量处理机

➤向量指令的源向量和目的向量都在存储器中，运算的中间结果需要送回存储器保存，对存储器的信息流量要求较高

6.3 向量处理方式-纵横处理方式

- 用于寄存器-寄存器结构的向量处理机中，向量寄存器的长度是有限的
- 当向量长度 N 大于向量寄存器长度 n 时，需要分组处理。
- 分组方法：

$$N = K \cdot n + r$$

其中： r 为余数，共分 $K + 1$ 组。组内采用纵向处理方式，组间采用横向处理方式。因此，也称为分组处理方式，纵横向加工方式等。

6.3 向量处理方式-纵横处理方式

➤ 运算过程为:

$$\text{第 1 组:} \quad T(1, n) = B(1, n) + C(1, n)$$

$$Y(1, n) = A(1, n) \times T(1, n)$$

$$\text{第 2 组:} \quad T(n+1, 2n) = B(n+1, 2n) + C(n+1, 2n)$$

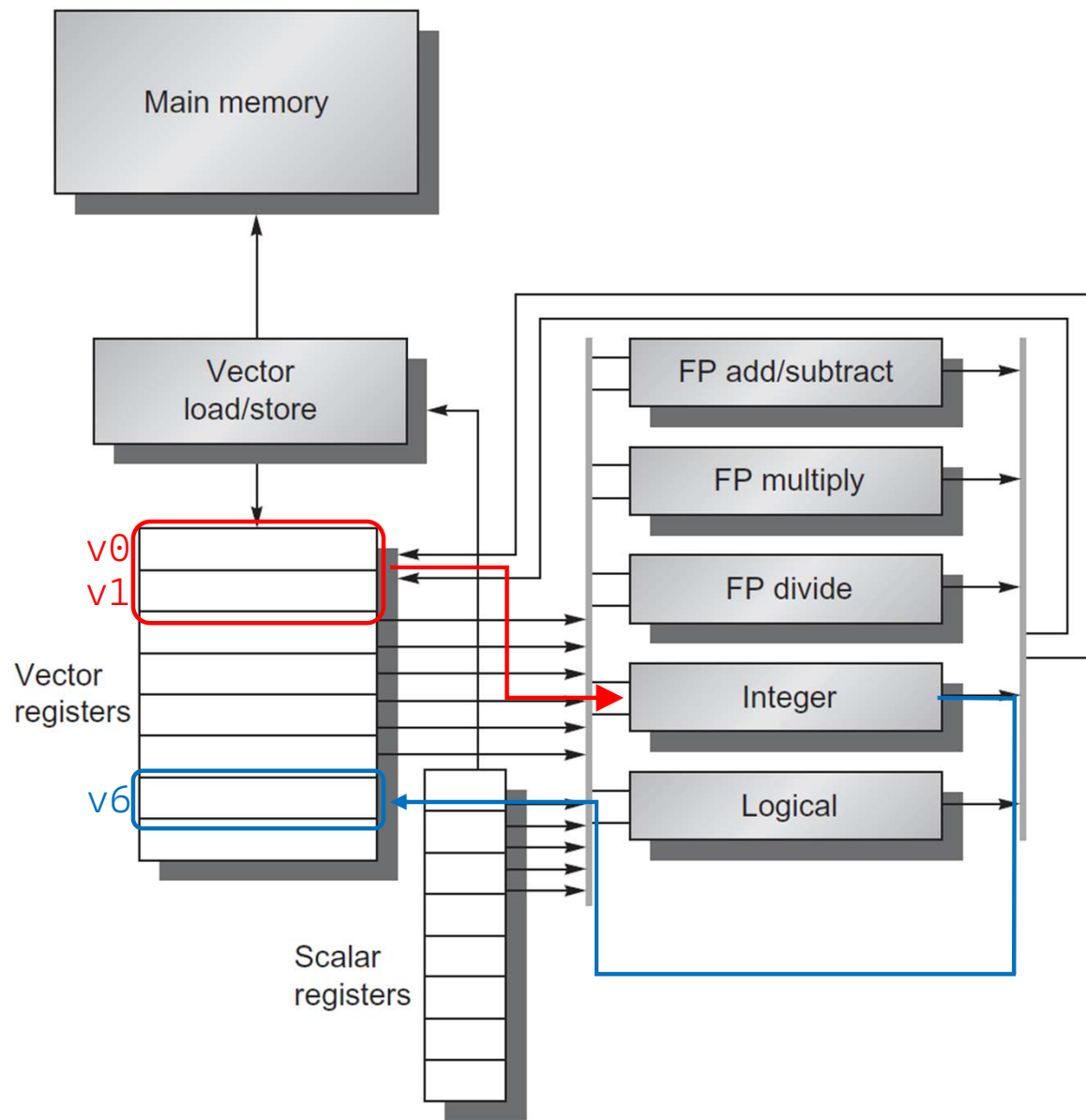
$$Y(n+1, 2n) = A(n+1, 2n) \times T(n+1, 2n)$$

.....

$$\text{最后第 } k+1 \text{ 组:} \quad T(kn+1, N) = B(kn+1, N) + C(kn+1, N)$$

$$Y(kn+1, N) = A(kn+1, N) \times T(kn+1, N)$$

➤ 主要优点: 在每组运算中, 用长度为 n 的向量寄存器作为运算寄存器并保留中间结果, 从而大大减少了访问存储器的次数, 提高了处理速度



vadd **v6**, **v0**, **v1**;

Figure 4.1 The basic structure of a vector architecture, RV64V, which includes a RISC-V scalar architecture. There are also 32 vector registers, and all the functional units are vector functional units. The vector and scalar registers have a significant number of read and write ports to allow multiple simultaneous vector operations. A set of crossbar switches (*thick gray lines*) connects these ports to the inputs and outputs of the vector functional units.

Mnemonic	Name	Description
vadd	ADD	Add elements of V[rs1] and V[rs2], then put each result in V[rd]
vsub	SUBtract	Subtract elements of V[rs2] from V[rs1], then put each result in V[rd]
vmul	MULTiply	Multiply elements of V[rs1] and V[rs2], then put each result in V[rd]
vdiv	DIVide	Divide elements of V[rs1] by V[rs2], then put each result in V[rd]
vrem	REMAinder	Take remainder of elements of V[rs1] by V[rs2], then put each result in V[rd]
vsqrt	SQUare ROoT	Take square root of elements of V[rs1], then put each result in V[rd]
vsll	Shift Left	Shift elements of V[rs1] left by V[rs2], then put each result in V[rd]
vsrl	Shift Right	Shift elements of V[rs1] right by V[rs2], then put each result in V[rd]
vsra	Shift Right Arithmetic	Shift elements of V[rs1] right by V[rs2] while extending sign bit, then put each result in V[rd]
vxor	XOR	Exclusive OR elements of V[rs1] and V[rs2], then put each result in V[rd]
vor	OR	Inclusive OR elements of V[rs1] and V[rs2], then put each result in V[rd]
vand	AND	Logical AND elements of V[rs1] and V[rs2], then put each result in V[rd]
vsgnj	SIGN source	Replace sign bits of V[rs1] with sign bits of V[rs2], then put each result in V[rd]
vsgnjn	Negative SIGN source	Replace sign bits of V[rs1] with complemented sign bits of V[rs2], then put each result in V[rd]
vsgnjx	Xor SIGN source	Replace sign bits of V[rs1] with xor of sign bits of V[rs1] and V[rs2], then put each result in V[rd]
vld	Load	Load vector register V[rd] from memory starting at address R[rs1]
vlds	Strided Load	Load V[rd] from address at R[rs1] with stride in R[rs2] (i.e., $R[rs1] + i \times R[rs2]$)
vldx	Indexed Load (Gather)	Load V[rs1] with vector whose elements are at $R[rs2] + V[rs2]$ (i.e., V[rs2] is an index)
vst	Store	Store vector register V[rd] into memory starting at address R[rs1]
vsts	Strided Store	Store V[rd] into memory at address R[rs1] with stride in R[rs2] (i.e., $R[rs1] + i \times R[rs2]$)
vstx	Indexed Store (Scatter)	Store V[rs1] into memory vector whose elements are at $R[rs2] + V[rs2]$ (i.e., V[rs2] is an index)
vpeq	Compare =	Compare elements of V[rs1] and V[rs2]. When equal, put a 1 in the corresponding 1-bit element of p[rd]; otherwise, put 0
vpne	Compare !=	Compare elements of V[rs1] and V[rs2]. When not equal, put a 1 in the corresponding 1-bit element of p[rd]; otherwise, put 0
vplt	Compare <	Compare elements of V[rs1] and V[rs2]. When less than, put a 1 in the corresponding 1-bit element of p[rd]; otherwise, put 0
vpxor	Predicate XOR	Exclusive OR 1-bit elements of p[rs1] and p[rs2], then put each result in p[rd]
vpor	Predicate OR	Inclusive OR 1-bit elements of p[rs1] and p[rs2], then put each result in p[rd]
vpand	Predicate AND	Logical AND 1-bit elements of p[rs1] and p[rs2], then put each result in p[rd]
setvl	Set Vector Length	Set vl and the destination register to the smaller of mvl and the source register

Figure 4.2 The RV64V vector instructions. All use the R instruction format. Each vector operation with two operands is shown with both operands being vector (.vv), but there are also versions where the second operand is a scalar register (.vs) and, when it makes a difference, where the first operand is a scalar register and the second is a vector register (.sv). The type and width of the operands are determined by configuring each vector register rather than being supplied by the instruction. In addition to the vector registers and predicate registers, there are two vector control and status registers (CSRs), vl and vctype, discussed below. The strided and indexed data transfers are also explained later. Once completed, RV64 will surely have more instructions, but the ones in this figure will be included.

6.4 向量处理机的关键技术

6.4.1 向量与标量性能的平衡

- 实际的应用问题中通常既有向量计算又有标量计算，而且两类计算有一定的比例
- 关键问题是：希望向量硬件和标量硬件都能够充分利用，不要空闲。
- 向量平衡点 (vector balance point)：为了使向量硬件设备和标量硬件设备的利用率相等，一个程序中向量代码所占的百分比。
 - 例如，一个系统的向量运算速度为90Mflops，标量运算速度为10Mflops。如果程序的90%是向量运算，10%是标量运算。则向量平衡点为0.9。硬件利用率最高。

几种超级计算机的向量性能和标量性能

机器型号	向量性能 Mflops	标量性能 Mflops	向量平衡点
Cray IS	85.0	9.8	0.90
Cray 2S	151.5	11.2	0.93
Cray X-MP	143.3	13.1	0.92
Cray Y-MP	201.6	17.0	0.92
Hitachi S820	737.3	17.8	0.98
NEC SX2	424.2	9.5	0.98
Fujitsu VP400	207.1	6.6	0.97

6.4 向量处理机的关键技术

6.4.2 向量链接技术

➤ 向量指令的类型：以CRAY-1向量处理机为例，有4类指令，2种指令格式

1) 向量与向量操作： $V_i \leftarrow V_j \text{ OP } V_k$

2) 向量与标量操作： $V_i \leftarrow S_j \text{ OP } V_k$

3) 向量取： $V_i \leftarrow \text{存储器}$

4) 向量存： $\text{存储器} \leftarrow V_i$

6.4 向量处理机的关键技术

6.4.2 向量链接技术

➤ 向量运算中的数据相关和功能部件冲突（采用顺序发射顺序完成方式）

(1) 写读数据相关。

(2) 读读数据相关，或向量寄存器冲突。

(3) 运算部件冲突。

$$V0 \leftarrow V1 + V2$$

$$V3 \leftarrow V4 \times V5$$

(a) 不相关的指令

$$V0 \leftarrow V1 + V2$$

$$V3 \leftarrow V4 + V5$$

(c) 功能部件冲突

$$V0 \leftarrow V1 + V2$$

$$V3 \leftarrow V0 \times V4$$

(b) 写读数据相关

$$V0 \leftarrow V1 + V2$$

$$V3 \leftarrow V1 \times V4$$

(d) 读读数据相关

6.4 向量处理机的关键技术

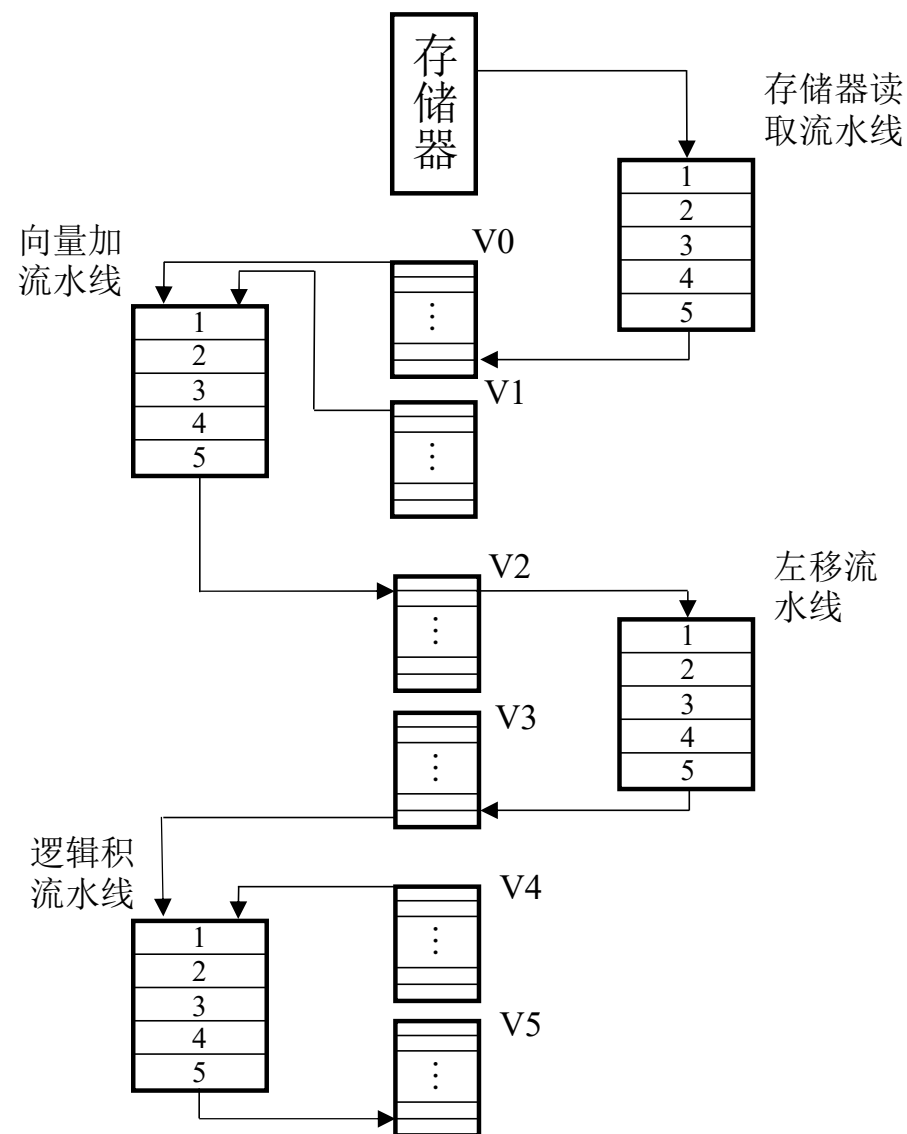
6.4.2 向量链接技术

- 当前一条指令的结果寄存器可以作为后继指令的操作数寄存器时，多条有数据相关的向量指令并行执行，这种技术称为两条流水线的链接技术。
- 当前一个向量功能部件产生第一个结果并送到结果向量寄存器入口时，将该结果立即送往下一个功能部件的入口，开始后续的向量操作。此后依次得到的中间结果都按此处理。这样，前面功能部件的结果元素一产生，就可以立即被后面的功能部件所使用，而不用等结果向量全部产生后再使用

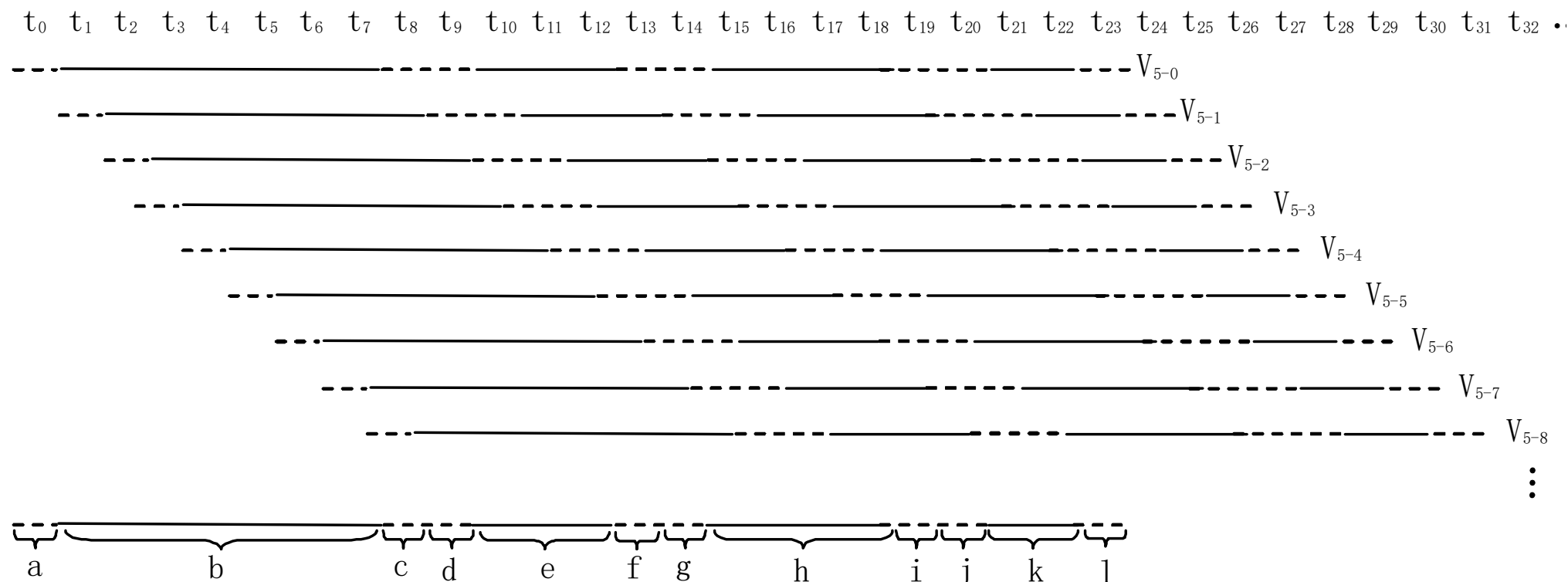
6.4 向量处理机的关键技术

6.4.2 向量链接技术

1: $V_0 \leftarrow \text{存储器}$
2: $V_2 \leftarrow V_0 + V_1$
3: $V_3 \leftarrow V_2 < 3$
4: $V_5 \leftarrow V_2 \wedge V_3$



链接操作的时间图:



a: 存储字到“读功能部件”的传送时间

c: 存储字从“读功能部件”到 V_0 分量的传送时间

e: 整数加功能部件的通过时间

g: V_2 中的操作数分量到移位功能部件的传送时间

i: 结果从移位功能部件到 V_3 分量的传送时间

k: 逻辑功能部件的通过时间

b: 存储字经过“读功能部件”的通过时间

d: V_0 和 V_1 中操作数到整数加功能部件的传送时间

f: 和从整数加功能部件到 V_2 分量的传送时间

h: 移位功能部件的通过时间

j: V_3 和 V_4 中的操作数分量到逻辑部件的传送时间

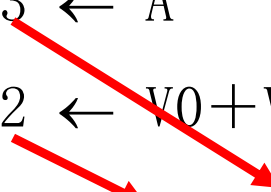
l: 最后结果到 V_5 分量的传送时间

6.4 向量处理机的关键技术

6.4.2 向量链接技术

例如：有如下3条向量指令：

1: $V3 \leftarrow A$
2: $V2 \leftarrow V0 + V1$
3: $V4 \leftarrow V2 \times V3$



- 第1、2条指令没有数据相关和功能部件冲突，可以同时开始执行。
- 第3条指令与第1、2条指令均存在写读数据相关，可以链接执行。

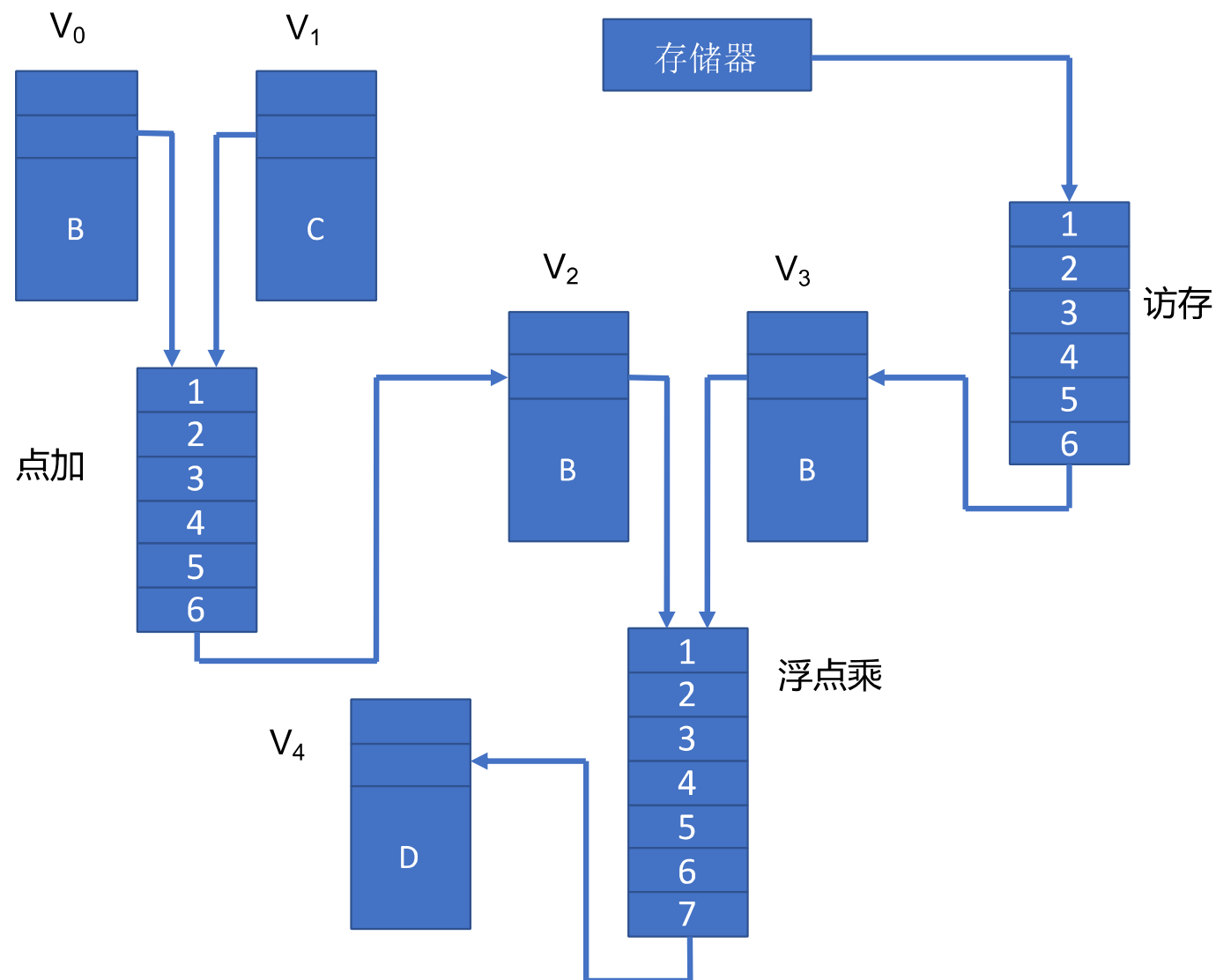
6.4 向量处理机的关键技术

6.4.2 向量链接技术

1: $V_3 \leftarrow A$

2: $V_2 \leftarrow V_0 + V_1$

3: $V_4 \leftarrow V_2 \times V_3$



6.4 向量处理机的关键技术

6.4.2 向量链接技术

向量链接的一些主要要求（除了要保证无向量寄存器使用冲突和无向量功能部件使用冲突的条件外）

- 在进行链接的时候，只有在前一条向量指令的**第一个结果元素**送入结果向量寄存器的那一个时钟周期才可以进行链接，若错过该时刻就不能进行链接，只有当前一条向量指令全部执行完毕，释放相应的向量寄存器资源后才能执行后面的向量指令
- 当一条向量指令的两个源操作数分别是两条先行向量指令的结果寄存器时，要求先行的两条向量指令产生运算**结果的时间**必须相等，即要求有关向量功能部件的延迟时间相等
- 只有所有可以链接执行的向量指令的**向量长度**相等时，它们之间才能链接执行，否则它们之间也不能链接执行

6.4 向量处理机的关键技术

6.4.2 向量链接技术

➤ 三种执行方式比较

1) 如果向量长度为N，三条指令采用串行方法执行的时间为：

$$[(1+6+1)+N-1]+[(1+6+1)+N-1]+[(1+7+1)+N-1]= 3N+22$$

2) 如果前两条指令并行执行，第三条指令串行执行，则执行时间

$$[(1+6+1)+N-1]+[(1+7+1)+N-1] = 2N+15$$

3) 采用向量链接技术，所需要拍数（即链接流水线的流水时间）为：

$$1\left\{\begin{array}{l}\text{启动访存} \\ \text{送浮加部件}\end{array}\right\}+6\left\{\begin{array}{l}\text{访存} \\ \text{浮加}\end{array}\right\}+1\left\{\begin{array}{l}\text{存}V_3 \\ \text{存}V_2\end{array}\right\}+1\left\{\begin{array}{l}\text{送浮乘部件} \\ \text{送浮乘部件}\end{array}\right\}+7\{\text{浮乘}\}+1\{\text{存}V_4\}=17$$

则三条向量指令总执行时间为： $17+(N-1)=N+16$

6.4 向量处理机的关键技术

6.4.3 向量循环开采技术

- 当向量的长度大于向量寄存器的长度时，必须把长向量分成长度固定的段，采用循环结构处理这个长向量，这种技术称为向量循环开采技术，也称为向量分段开采技术。

A和B为长度N的向量。

```
for (i = 1; i < N; i++)
```

```
    A[i] = 5 * B[i] + C;
```

当向量长度N为64或更小时，计算A数组的7条指令序列是：

向量寄存器长度

- | | |
|-----------------------------------|-----------------------|
| 1: $S_1 \leftarrow 5.0$ | 在标量寄存器内设置常数 |
| 2: $S_2 \leftarrow C$ | 将常数C装入标量寄存器 |
| 3: $VL \leftarrow N$ | 在VL寄存器内设置向量长度 |
| 4: $V_0 \leftarrow B$ | 将B向量读入向量寄存器 |
| 5: $V1 \leftarrow S_1 \times V_0$ | B数组的每个分量乘常数 |
| 6: $V2 \leftarrow S_2 + V1$ | C和 $5 \times B(x)$ 相加 |
| 7: $A \leftarrow V2$ | 将结果向量存入A数组 |

当N超过64时，要采用向量循环开采技术。

在进入循环前，把N除以64，确定循环次数。

第4条到第7条指令组成循环

1: $S_1 \leftarrow 5.0$ 在标量寄存器内设置常数

2: $S_2 \leftarrow C$ 将常数C装入标量寄存器

for ($i=0; i \geq N/64; i++$) {

3: $VL \leftarrow \min(N, 64)$ 在VL寄存器内设置向量长度

4: $V_0 \leftarrow B$ 将B向量读入向量寄存器

5: $V1 \leftarrow S_1 \times V_0$ B数组的每个分量乘常数

6: $V2 \leftarrow S_2 + V1$ C和 $5 \times B(x)$ 相加

7: $A \leftarrow V2$ 将结果向量存入A数组

8: $N \leftarrow N - \min(64, N)$

}

6.4 向量处理机的关键技术

6.4.4 向量递归技术

- 向量指令可以通过让源向量和结果向量使用同一个向量寄存器组，并控制移出和移入分量计数器的修改，实现递归操作
 - 在流水线的每一个周期，当一个操作数分量移出向量寄存器进入流水线功能部件时，一个分量结果可以在同一周期进入腾空的分量寄存器
 - 分量计数器必须跟踪位移操作，直到结果向量的所有64个分量都装入向量寄存器

6.4 向量处理机的关键技术

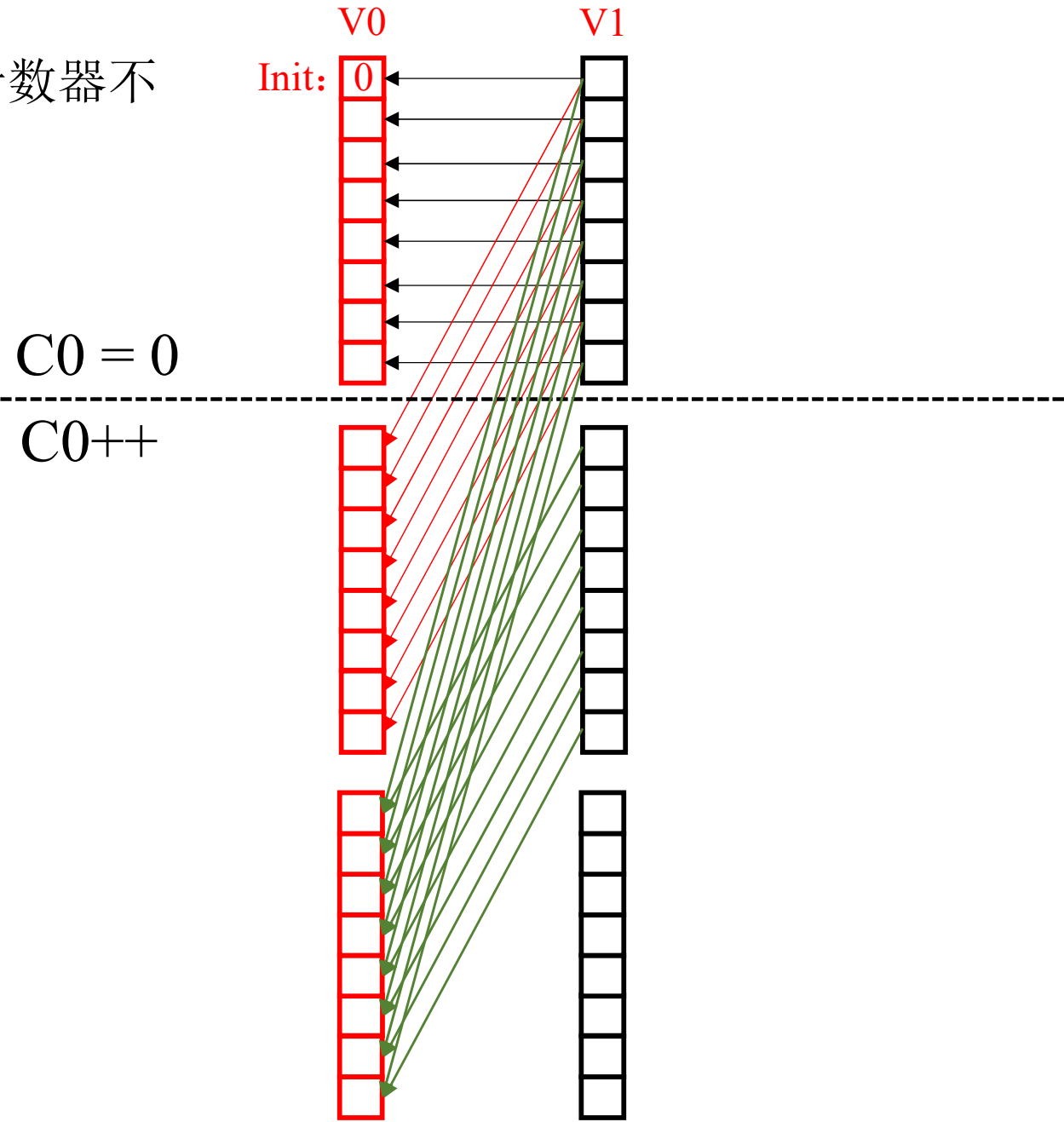
6.4.4 向量递归技术

- 实现向量各分量有依赖关系的聚合操作， $SUM(v_i)$
- 利用浮点加法流水线完成递归向量求和

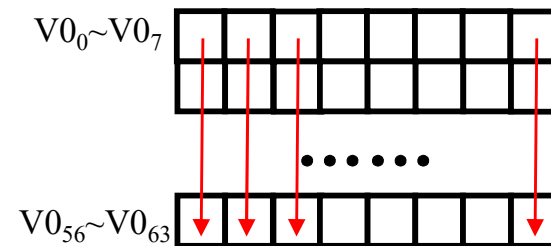
$$V0 \leftarrow V0 + V1$$

- 向量寄存器V1保存要进行递归相加的浮点数，向量寄存器V0同时用作操作数寄存器和结果寄存器
- C0和C1分别是与向量寄存器V0和V1相关的移入分量计数器。初始时，计数器C0和C1都置成0， $V0_0$ 中的初始值也置成0。
- 浮点加法流水线的延迟时间为8个周期。移入移出相隔8个是安全的。
- 假定向量长度为64，只作一个向量循环。
- 在开始的8个周期，计数器C0一直为0，在此之后，每个周期期加1。C1每个周期加1。

写数据分量计数器不
受影响



$$\left. \begin{aligned} V0_0 &= V0_0 + V1_0 = 0 + V1_0 \\ V0_1 &= V0_0 + V1_1 = 0 + V1_1 \\ &\dots\dots \\ V0_7 &= V0_0 + V1_7 = 0 + V1_7 \end{aligned} \right\} \text{第1次加法}$$



$$\left. \begin{aligned} V0_8 &= V0_0 + V1_8 = V1_0 + V1_8 \\ &\dots\dots \\ V0_{15} &= V0_7 + V1_{15} = V1_7 + V1_{15} \\ V0_{16} &= V0_8 + V1_{16} = V1_0 + V1_8 + V1_{16} \\ &\dots\dots \end{aligned} \right\} \text{第2次加法}$$

$$\left. \begin{aligned} V0_{56} &= V0_{48} + V1_{56} = V1_0 + V1_8 + V1_{16} + V1_{24} + V1_{32} + V1_{40} + V1_{48} + V1_{56} \\ &\dots\dots \\ V0_{63} &= V0_{55} + V1_{63} = V1_7 + V1_{15} + V1_{23} + V1_{31} + V1_{39} + V1_{47} + V1_{55} + V1_{63} \end{aligned} \right\} \text{第8次加法}$$

经过8次运算, 得到8个结果, 分别是8个数的和



6.4 向量处理机的关键技术

6.4.4 向量递归技术

➤ 向量点积 $A \cdot B = \Sigma(a_i \cdot b_i)$

- $a_i \cdot b_i$ 用普通向量指令求得，存于 $C = \{c_i\}$
- Σc_i 存在数据相关
- 向量递归技术：64 \rightarrow 8;

6.5 向量处理机实例

6.5.1 典型向量处理机

6.5.2 CRAY Y-MP向量处理机

6.5.3 向量协处理器

6.5.1 典型的向量处理机

➤ 向量处理机主要出自美国和日本

➤ 美国著名的向量计算机公司有：CRAY、CDC、TI等

➤ 日本公司有：NEC、Fujitsu、Hitachi等

系统型号	推出时间	最大配置, 时钟周期, 操作系统/编译系统	特色和要点
Cray 1S	1976年	有10条流水线的单处理机, 12.5ns, COS/CF7 2.1	第一台基于ECL的超级计算机
Cray 2S/4-256	1985年	256M字存储器的4台处理机, 4.1ns, COS或UNIX/CF77 3.0	16K字的本地存储器, 移植了UNIX V
Cray X-MP 416	1983年	16M字存储器的4台处理机, 128M字 SSD, 8.5ns, COS/CF77 5.0	使用共享寄存器组用于 IPC
Cray Y-MP 832	1988年	128M字存储器的8台处理机, 6ns, CF77 5.0	X-MP的改进型
Cray Y-MP C-90	1991年	每台处理机2条向量流水线, 16台处理 机, 4.2ns, UNICOS/CF77 5.0	最大的Cray机器
CDC Cyber 205	1982年	有4条流水线的单处理机, 20ns, 虚拟OS/FTN200	存储器-存储器系统结 构

6.5.1 典型的向量处理机

➤ 向量处理机主要出自美国和日本

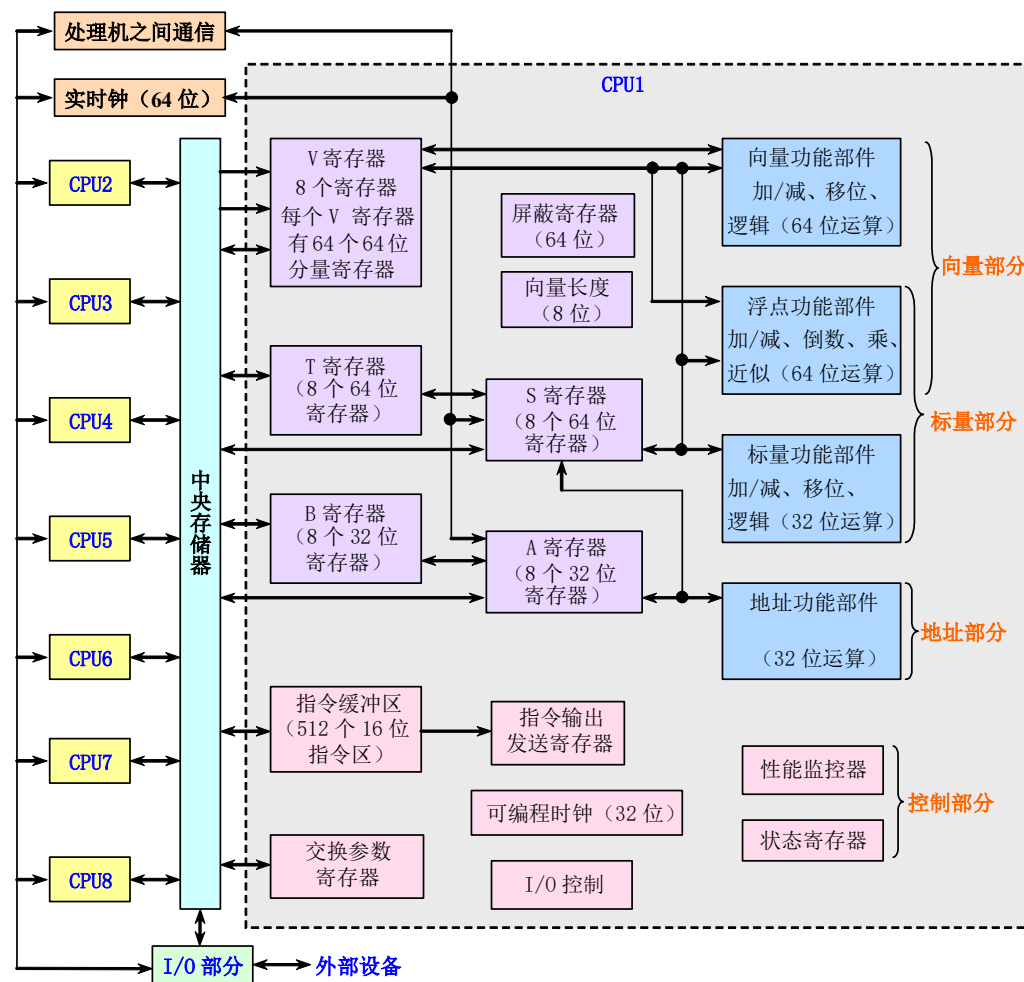
➤ 美国著名的向量计算机公司有：CRAY、CDC、TI等

➤ 日本公司有：NEC、Fujitsu、Hitachi等

系统型号	推出时间	最大配置, 时钟周期, 操作系统/编译系统	特色和要点
ETA 10E	1985年	单处理机, 10.5ns, ETAV/FTN 200	Cyber 205的后继型号
NEC SX-X/44	1991年	每台处理机4组流水线, 4台处理机, 2.9ns, F77SX	
Fujitsu VP2600/10	1991年	5条流水线的单处理机和双标量处理 机, 3.2ns, MSP.EX/F77 EX/VP	使用可重构微向量寄 存器和屏蔽
Hitachi 820/80	1988年	512MB存储器, 18个流水线功能部 件的单处理机, 4ns, FORT 77/HAP V23-OC	64个I/O通道, 最大传 输率为288MB/秒

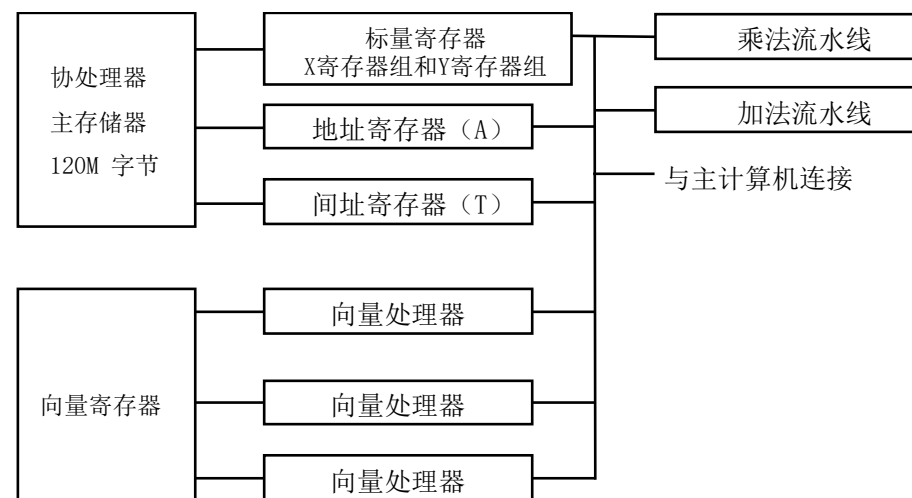
6.5.2 CRAY Y-MP向量处理机

- 由1至8个处理机组成，共享中央存储器、I/O子系统、处理机通信子系统和实时钟。
- 中央存储器由256个交叉访问的存储体组成。每个处理机对4个存储器端口交叉访问。
- CPU的时钟周期为6ns。
- 每个CPU由14个功能部件组成，分为向量、标量、地址和控制四个子系统。
- 使用了大量地址寄存器、标量寄存器、向量寄存器、中间寄存器和临时寄存器。
- 可以实现功能流水线灵活的链接。
- I/O子系统支持三类通道，传输速率分别为6兆字节/秒，100兆字节/秒和1G字节/秒。



6.5.3 向量协处理器

- 以中小型机或微机作主机，向量处理部件作为外围设备，加速向量的处理速度。
- 向量协处理器是为中小型用户设计的，解决科学计算中大量向量处理任务的一种装置。
- FPS-164是最典型的向量协处理器，美国浮点系统公司生产。每个向量处理器有两个乘加部件，两组向量寄存器，两组标量寄存器。向量寄存器有2组 \times 4个 \times 2 K个操作数，每个操作数4个字节。
- 各向量处理器同步地运算，但它们处理的数据各不相同。
- 向量操作可以和标量处理器中的标量操作同时进行



6.6 向量处理机的性能评价

衡量向量处理机性能的主要指标有：

- 向量指令处理时间 T_{vp}
- 最大性能 R_{∞}
- 半性能向量长度 $n_{1/2}$
- 向量长度临界值 n_v

6.6.1 向量指令处理时间

1. 一条向量指令处理时间

➤ 执行一条长度为 n 的向量指令的时间 T_{vp} 表示为：

$$T_{vp} = T_s + T_{vf} + (n - 1)T_c$$

➤ T_s 为向量流水线的建立时间, 即为了使处理部件流水线能开始工作（即开始流入数据）所需要的准备时间

➤ T_{vf} 为向量流水线的流过时间, 即第一对向量元素通过流水线并产生第一个结果所花的时间

➤ T_c 为流水线的时钟周期时间。

6.6.1 向量指令处理时间

1. 一条向量指令处理时间

➤ 执行一条长度为 n 的向量指令的时间 T_{vp} 表示为：

$$T_{vp} = T_s + T_{vf} + (n - 1)T_c$$

➤ 将上式中的参数都折算成时钟周期个数，则有：

$$T_{vp} = [s + e + (n - 1)]\tau$$

- s 为向量流水线建立所需的时钟周期数，
- e 为向量流水线流过所需的时钟周期数，
- n 为向量长度，
- τ 为时钟周期长度

6.6.1 向量指令处理时间

1. 一条向量指令处理时间

➤ 执行一条长度为 n 的向量指令的时间 T_{vp} 表示为：

$$T_{vp} = T_s + T_{vf} + (n - 1)T_c$$

➤ 将上式中的参数都折算成时钟周期个数，则有：

$$T_{vp} = [s + e + (n - 1)]\tau$$

➤ 不考虑 T_s ，并令 $T_{start} = e - 1$ 则有：

$$T_{vp} = [T_{start} + n]\tau$$

➤ T_{start} ：从第一条向量指令开始执行，到还差一个时钟周期就产生第一个结果所需要的时钟周期数。可称之为该向量指令的启动时间。此后，每个时钟周期流出一个结果，共有 n 个结果

6.6.1 向量指令处理时间

2. 一组向量指令的处理时间

➤ 一组向量操作的执行时间主要取决于

- 向量长度
- 向量操作之间是否存在流水线功能部件的冲突
- 数据的相关性

➤ 将几条能在一个时钟周期内一起开始执行的向量指令称为一个编队

- 同一个编队中的向量指令一定不存在流水部件的冲突和数据相关性
- 如果存在部件冲突和数据相关，需要把它们分在不同的编队中

6.6.1 向量指令处理时间

2. 一组向量指令的处理时间

- 一组向量操作的执行时间主要取决于：向量长度、向量操作之间是否存在流水线功能部件的冲突、数据的相关性
- 将几条能在一个时钟周期内一起开始执行的向量指令称为一个编队
- 编队后，向量指令序列的总的执行时间为 m 个编队的执行时间的和

$$T_{all} = \sum_{i=1}^m T_{vp}^{(i)}$$

- $T_{vp}^{(i)}$ ：第 i 个编队的执行时间
- m ：编队的个数

6.6.1 向量指令处理时间

2. 一组向量指令的处理时间

- 一组向量操作的执行时间主要取决于：**向量长度、向量操作之间是否存在流水线功能部件的冲突、数据的相关性**
- 将几条能在一个时钟周期内一起开始执行的向量指令称为一个编队
- 编队后，向量指令序列的总的执行时间为 m 个编队的执行时间的和 $T_{all} = \sum_{i=1}^m T_{vp}^{(i)}$
- 当一个编队是由若干条指令组成时，其执行时间就应该由该编队中各指令的执行时间的最大值来确定

$$T_{all} = \sum_{i=1}^m T_{vp}^{(i)} = \sum_{i=1}^m (T_{start}^{(i)} + n) T_c = \left(\sum_{i=1}^m T_{start}^{(i)} + mn \right) T_c = (T_{start} + mn) T_c$$

- $T_{start}^{(i)}$: 第 i 个编队中各指令的启动时间的最大值
- $T_{start} = \sum_{i=1}^m T_{start}^{(i)}$: 该组指令总的启动时间（时钟周期个数）

6.6.1 向量指令处理时间

2. 一组向量指令的处理时间

- 一组向量操作的执行时间主要取决于：向量长度、向量操作之间是否存在流水线功能部件的冲突、数据的相关性
- 将几条能在一个时钟周期内一起开始执行的向量指令称为一个编队
- 编队后，向量指令序列的总的执行时间为 m 个编队的执行时间的和 $T_{all} = \sum_{i=1}^m T_{vp}^{(i)}$
- 当一个编队是由若干条指令组成时，其执行时间就应该由该编队中各指令的执行时间的最大值来确定

$$T_{all} = \sum_{i=1}^m T_{vp}^{(i)} = \sum_{i=1}^m (T_{start}^{(i)} + n) T_c = \left(\sum_{i=1}^m T_{start}^{(i)} + mn \right) T_c = (T_{start} + mn) T_c$$

- 表示成时钟周期个数

$$T_{all} = T_{start} + mn$$

假设每种向量功能部件只有1个，且不考虑向量链接，那么下面一组向量指令能分成几个编队？

LV V1, Rx ; 取向量x

MULTSV V2, F0, V1 ; 向量和标量相乘

LV V3, Ry ; 取向量Y

ADDV V4, V2, V3 ; 加法

SV Ry, V4 ; 存结果

假设每种向量功能部件只有1个，且不考虑向量链接，那么下面一组向量指令能分成几个编队？

Diagram illustrating a sequence of five vector instructions with data dependencies indicated by red arrows:

- LV V1, Rx ; 取向量x
- MULTSV V2, F0, V1 ; 向量和标量相乘
- LV V3, Ry ; 取向量Y
- ADDV V4, V2, V3 ; 加法
- SV Ry, V4 ; 存结果

Red arrows show dependencies: V1 is used in MULTSV; V2 is used in ADDV; V3 is used in ADDV; V4 is used in SV.

假设每种向量功能部件只有1个，且不考虑向量链接，那么下面一组向量指令能分成几个编队？

LV V1, Rx ; 取向量x

MULTSV V2, F0, V1 ; 向量和标量相乘

LV V3, Ry ; 取向量Y

ADDV V4, V2, V3 ; 加法

SV Ry, V4 ; 存结果

6.6.1 向量指令处理时间

3. 分段时一组向量指令的总执行时间

- 如果考虑向量长度 n 大于向量寄存器长度 MVL 时, 则需要分段开采。
- 执行模型: 一段数据完成整组向量指令
- 引入一些额外的处理操作
 - 假设这些额外操作所引入的额外时间为 T_{loop} 个时钟周期
 - 设 $\left\lfloor \frac{n}{MVL} \right\rfloor = p, q$ 为余数
 - 共有 m 个编队
- 对于最后一次循环来说, 所需要的时间为

$$T_{last} = T_{start} + T_{loop} + m \times q$$

6.6.1 向量指令处理时间

3. 分段时一组向量指令的总执行时间

➤ 其它的每一次循环所需要花费的时间为

$$T_{step} = T_{start} + T_{loop} + m \times MVL$$

➤ 总的执行时间为

$$\begin{aligned} T_{all} &= T_{step} \times p + T_{last} \\ &= (T_{start} + T_{loop} + m \times MVL) \times p + (T_{start} + T_{loop} + m \times q) \\ &= (p + 1) \times (T_{start} + T_{loop}) + m(MVL \times p + q) \\ &= \left\lceil \frac{n}{MVL} \right\rceil \times (T_{start} + T_{loop}) + mn \end{aligned}$$

在某台向量处理机上的执行DAXPY的向量指令序列，即完成

$$Y = a \times X + Y$$

其中， X 和 Y 是向量，最初保存在主存中， a 是标量，已经存放在寄存器F0中。向量指令序列如下

- 1: LV V1, Rx ;取向量x
- 2: MULTSV V2, F0, V1 ;向量和标量相乘
- 3: LV V3, Ry ;取向量Y
- 4: ADDV V4, V2, V3 ;加法
- 5: SV Ry, V4 ;存结果

假设向量寄存器长度 $MVL = 64$ ， $T_{loop} = 15$ ，各功能部件的启动时间为

- 取数和存数部件为12个时钟周期
- 乘法部件为7个时钟周期
- 加法部件为6个时钟周期

分别完成对于不采用向量链接技术和采用向量链接技术的两种情况，求完成上述向量操作的执行时间

当不采用向量链接技术时，将上述5条向量指令分成4个编队

- | | | |
|-----------|------------|----------|
| 1: LV | V1, Rx | ;取向量x |
| 2: MULTSV | V2, F0, V1 | ;向量和标量相乘 |
| 3: LV | V3, Ry | ;取向量Y |
| 4: ADDV | V4, V2, V3 | ;加法 |
| 5: SV | Ry, V4 | ;存结果 |

则有

$$T_{start} = 12 + 12 + 6 + 12 = 42, m = 4$$

那么，对 n 个向量元素进行DAXPY表达式计算所需要的时钟周期个数为

$$\begin{aligned} T_{all} &= \left\lceil \frac{n}{MVL} \right\rceil \times (T_{start} + T_{loop}) + mn \\ &= \left\lceil \frac{n}{64} \right\rceil \times (42 + 15) + 4n \\ &= \left\lceil \frac{n}{64} \right\rceil \times 57 + 4n \end{aligned}$$

若采用向量链接技术，那么上述5条向量指令的编队结果如下
($m = 3$)

- 第一编队：LV V1, RX; MULTFV V2, F0, V1;
- 第二编队：LV V3, Ry; ADDV V4, V2, V3;
- 第三编队：SV V4, Ry;

前两个编队中分别是取数和乘法指令链接以及取数和加法指令链接，可知

- 第一编队启动需要 $12+7=19$ 个时钟周期
- 第二编队启动需要 $12+6=18$ 个时钟周期
- 第三编队启动需要12个时钟周期

那么，对 n 个向量元素进行DAXPY表达式计算所需要的时钟周期个数为

$$\begin{aligned} T_{all} &= \left\lceil \frac{n}{MVL} \right\rceil \times (T_{start} + T_{loop}) + mn \\ &= \left\lceil \frac{n}{64} \right\rceil \times (19 + 18 + 12 + 15) + 3n \\ &= \left\lceil \frac{n}{64} \right\rceil \times 64 + 3n \end{aligned}$$

在一台向量处理机上实现 $A=B \times s$ 操作，其中A和B是长度为200的向量，s是一个标量。向量寄存器长度为64。各功能部件的启动时间与上例相同。求不采用向量链接技术时，每个元素的平均处理时间。

$$T_{\text{loop}}=15; \text{ 存取数: } 12; \text{ 向量乘: } 7$$

因为向量长度超过了向量寄存器的长度，所以要采取分段开采方法。假设A和B分别放在Ra和Rb之中，s放在Fs中。每次循环主要由下面三条向量指令组成：

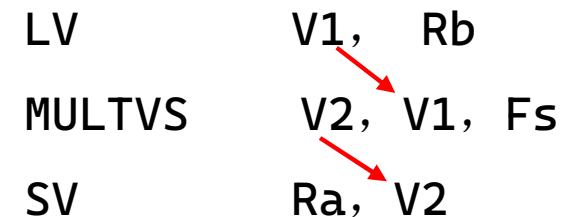
LV	V1, Rb	;取向量B
MULTVS	V2, V1, Fs	;向量和标量相乘
SV	Ra, V2	;存向量

- 三条指令之间存在有写读数据相关，因此必须把它们分成 $m = 3$ 个编队

$$\begin{aligned}
 T_{all} &= \left\lceil \frac{n}{MVL} \right\rceil \times (T_{start} + T_{loop}) + 200 \times 3 \\
 &= \left\lceil \frac{200}{64} \right\rceil \times (12 + 7 + 12 + 15) + 600 \\
 &= 784
 \end{aligned}$$

- 每个结果元素的平均执行时间为：

$$\frac{784}{200} = 3.9$$



6.6.2 最大性能 R_{∞} 和半性能向量长度 $n_{1/2}$

1. 向量处理机的峰值性能 R_{∞}

- R_{∞} 表示当向量长度为无穷大时，向量处理机的最高性能，也称为峰值性能，单位是MFLOPS

$$R_{\infty} = \lim_{n \rightarrow \infty} \frac{\text{向量执行序列中浮点运算次数} \times \text{时钟频率}}{\text{向量指令序列执行所花费的时钟周期数}}$$

每个时钟周期所能完成的浮点运算次数

在某台向量处理机上的执行DAXPY的向量指令序列，即完成

$$Y = a \times X + Y$$

其中， X 和 Y 是向量，最初保存在主存中， a 是标量，已经存放在寄存器F0中。向量指令序列如下

- | | | |
|-----------|------------|----------|
| 1: LV | V1, Rx | ;取向量x |
| 2: MULTSV | V2, F0, V1 | ;向量和标量相乘 |
| 3: LV | V3, Ry | ;取向量Y |
| 4: ADDV | V4, V2, V3 | ;加法 |
| 5: SV | Ry, V4 | ;存结果 |

假设该向量处理机的时钟频率为200MHz

$$\begin{aligned} R_{\infty} &= \lim_{n \rightarrow \infty} \frac{\text{向量执行序列中浮点运算次数} \times \text{时钟频率}}{\text{向量指令序列执行所花费的时钟周期数}} \\ &= \lim_{n \rightarrow \infty} \frac{2 \times n \times 200}{\left\lceil \frac{n}{64} \right\rceil \times 64 + 3n} \\ &= 100 \text{ MFLOPS} \end{aligned}$$

6.6.2 最大性能 R_∞ 和半性能向量长度 $n_{1/2}$

2. 半性能向量长度 $n_{1/2}$

- 半性能向量长度 $n_{1/2}$ 是指向量处理机的性能为其最大性能的一半所需要的向量长度
- 评价向量流水线的建立时间对性能影响的重要参数

$$\frac{2 \times n_{1/2} \times 200}{\left\lceil \frac{n_{1/2}}{64} \right\rceil \times 64 + 3n_{1/2}} = 50$$

假设 $n_{1/2} \leq 64$, 则 $n_{1/2} = 13$

$$\frac{2 \times n_{1/2} \times 200}{\left\lceil \frac{n_{1/2}}{64} \right\rceil \times 64 + 3n_{1/2}} = 50$$

$$5 \times n_{1/2} - 64 = 0 \quad n_{1/2} \leq 64$$

$$5 \times n_{1/2} - 128 = 0 \quad 64 < n_{1/2} \leq 128$$

....

$$8 \times n_{1/2} = \left\lceil \frac{n_{1/2}}{64} \right\rceil \times 64 + 3n_{1/2}$$

$$5 \times n_{1/2} - \left\lceil \frac{n_{1/2}}{64} \right\rceil \times 64 = 0$$

6.6.2 最大性能 R_∞ 和半性能向量长度 $n_{1/2}$

3. 向量长度临界值

- 向量长度临界值 n_v 是指：对于某一计算任务而言，向量方式的处理速度优于标量串行方式处理速度时所需要的最小向量长度
- 对于上述例子
 - 假设在标量串行工作方式下实现DAXPY循环的开销为10个时钟周期，则在该方式下，计算DAXPY所需要的时钟周期为

$$T_s = (10 + 12 + 12 + 7 + 6 + 12) \times n_v = 59n_v$$

- 在向量方式下，计算DAXPY所需要的时钟周期数为

$$T_v = 64 + 3n_v$$

- 根据向量长度临界值的定义，有 $T_v = T_s$

$$n_v = \left\lceil \frac{64}{56} \right\rceil = 2$$

6.7 向量处理机的发展

1. 向量计算机系统结构解决的六个技术问题：

- 1) 处理机带宽，有两条途径：运算部件采用流水线结构、用多个运算器构成并行系统。
- 2) 存储器带宽，多种解决方法：
 - 用多个独立的存储体构造大容量存储器系统。
 - 采用多层次的存储器系统提高访问速度。
 - 采用Cache和可寻址的寄存器组效果最好。
 - 采用流水线技术，存储系统的速度快5~20倍
- 3) 输入 / 输出带宽，有10~29个DMA通道。
- 4) 通信带宽，共享存储器或互连网络。
- 5) 同步系统，多流水线结构通过控制程序使所有流水线能够同步工作。
- 6) 多用途，例如，能够处理非数值计算问题

6.7 向量处理机的发展

2. 向量计算机系统结构的发展趋势是：

- 1) 提供多种向量运算指令。
- 2) 除具有向量处理功能外还有其它功能。
- 3) 采用多层次的存储器系统。
- 4) 流水线技术与并行技术相结合。

本章重点：

1. 向量的表示方法
2. 向量运算中的相关性与向量链接技术
3. 向量递归技术
4. 向量处理机的性能评价