计算机系统结构

第13章 阵列处理机

目录

引言

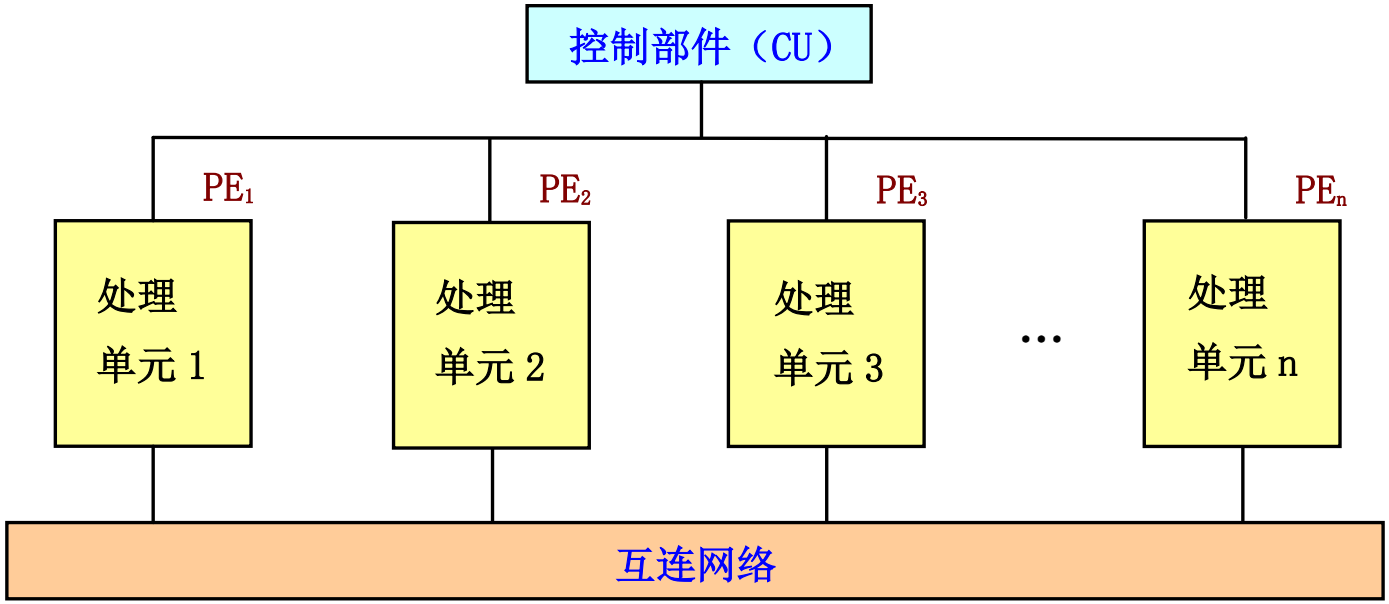
阵列处理机

- 核心：一个由多个处理单元构成的阵列
- 采用资源重复的方法，设置较多的处理单元来提高并行性。
- 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。
  - 又称为**SIMD计算机（单指令流多数据流）**。
- 有时还被称为并行处理机。

13.1 阵列处理机的操作模型和特点

1. 阵列处理机的操作模型

- 用一个控制部件CU同时管理多个处理单元PE。
- CU对指令进行译码，并把指令播送到各处理单元。
- 所有处理单元均被动地接收并执行从控制部件广播来的同一条指令，但它们所操作的对象却是不同的数据。



阵列处理机的操作模型

2. 阵列处理机的操作模型可用五元组表示

阵列处理机＝（N，C，I，M，R）

- N：机器的处理单元（PE）数。

  例如：Illiac IV计算机有64个PE；MP-1计算机有16384个PE

- C：控制部件CU直接执行的指令集，包括标量指令和程序流控制指令。
- I：由CU广播至所有PE进行并行执行的指令集。
  - 包括算术运算、逻辑运算、数据寻径、屏蔽以及其他由每个PE对它的数据所执行的局部操作。
- M：屏蔽方案集
  - 每种屏蔽将所有PE划分成允许操作和禁止操作两种工作模式。
- R：数据寻径功能集
  - 说明互连网络中PE间通信所需要的各种设置模式。

例如：MasPar MP-1计算机的操作特性如下：

（1）【N】MP-1是一种SIMD机器，其PE数 N＝1024～16384。

（2）【C】CU执行标量指令，将译码后的向量指令广播到PE阵列，并控制PE间通信。

（3）【I】每个PE都是RISC处理机，能执行不同数据的整数运算和标准浮点运算。PE从CU接收指令。

（4）【M】屏蔽方案设在每个PE中，并由CU连续监控，它能在运行时动态地使每个PE处于工作或禁止状态。

（5）【R】 MP-1有一个X-Net网格网络和一个全局多级交叉开关寻径器，以实现CU-PE之间、X-Net的8个近邻和全局寻径器的通信。

2. 阵列处理机的特点

- 以单指令流多数据流方式工作（SIMD）。
- 通过设置多个相同的处理单元来开发并行性。
  - 利用并行性中的同时性，而不是并发性。所有处理单元必须同时进行相同的操作。
- 以某一类算法为背景的专用计算机。
- 阵列机的研究必须与并行算法的研究密切结合，以便能充分发挥它的处理能力。
- 阵列机的控制器实质上是一台标量处理机，而为了完成I/O操作以及操作系统的管理，尚需一个前端机。

实际的阵列机系统是由3部分构成的一个异构型多处理机系统：

PE+控制器+前端机

13.2 阵列处理机的基本结构

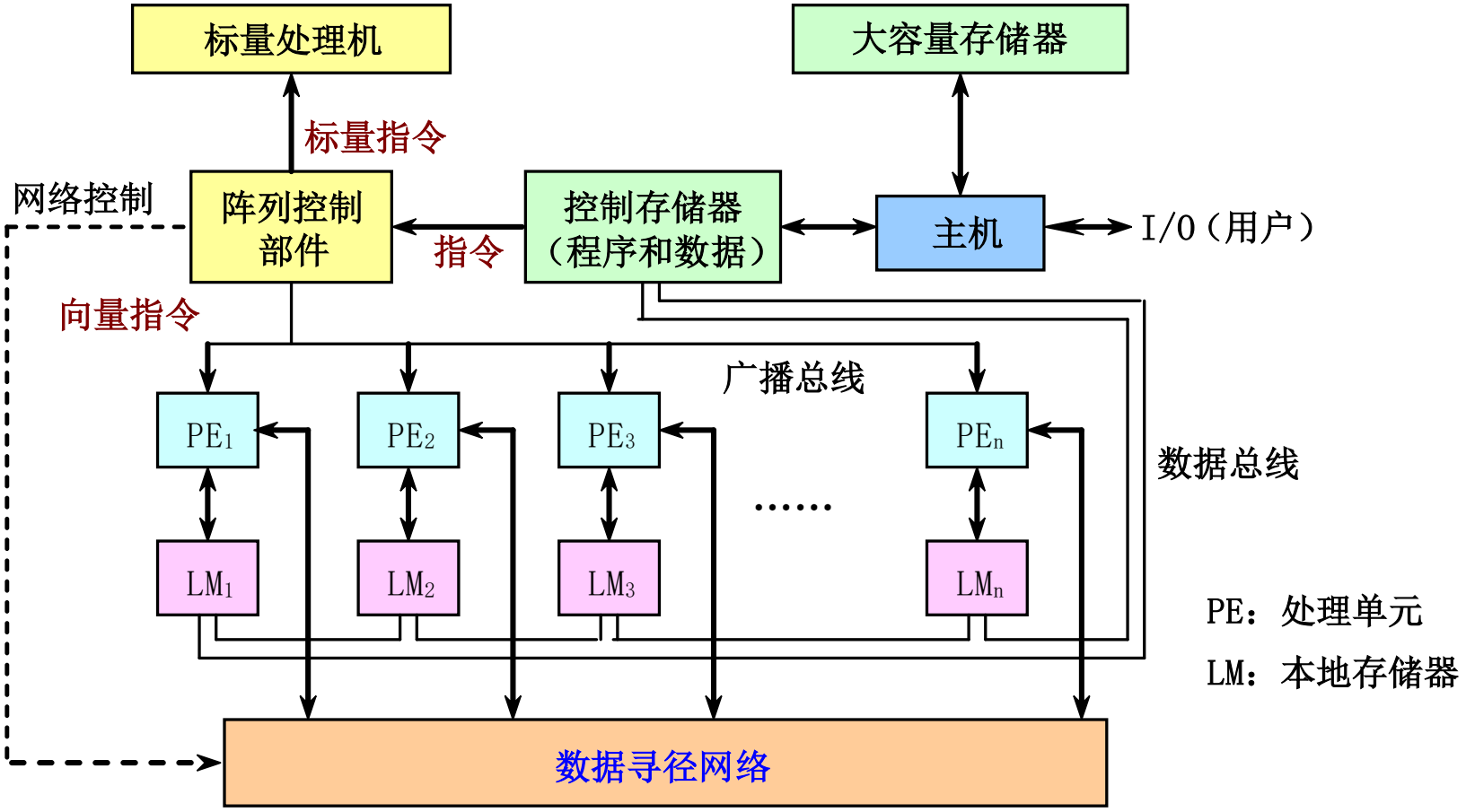
13. 2. 1 分布式存储器的阵列机

1. 分布式存储器的阵列机结构

- 含有多个相同的处理单元PE，每个PE有各自的本地存储器LM。
- PE之间通过数据寻径网络以一定方式互相连接。它们在阵列控制部件的统一指挥下，实现并行操作。
- 指令的执行顺序基本上是串行进行的。
- 程序和数据是通过主机装入控制存储器。

数据寻径网络：执行PE间的通信，如移数、置换和其它寻径操作的互联网络。控制部件通过执行程序来控制数据寻径网络。PE的同步由控制部件的硬件实现。

13.2 阵列处理机的基本结构



分布式存储器的阵列处理机结构

2. 指令送到控制部件进行译码。
   - 标量指令：直接由标量处理机执行。
   - 向量指令：阵列控制部件通过广播总线将它广播到所有PE中去并行地执行。
3. 执行程序所需的数据集经划分后通过数据总线分布存放到各PE的本地存储器LM。
4. 各PE之间通过数据寻径网络互连，实现PE间的通信，控制部件通过执行程序来控制数据寻径网络。
5. PE的同步是在控制部件的控制下由硬件实现。
   - 可以让所有PE在同一个周期执行同一条指令
   - 也可以通过采用屏蔽逻辑来控制某些PE在指定的指令周期是否参与执行

6. 各种阵列处理机的主要差别在于数据寻径网络的不同。

- Illiac IV： 4-邻连接网络结构
  （在过去是最常用的一种）
- CM-2：嵌在网格中的超立方体
- MasPar MP-1：X-Net加多级交叉开关寻径器

13. 2. 2 共享存储器的阵列机

共享存储器的阵列处理机结构

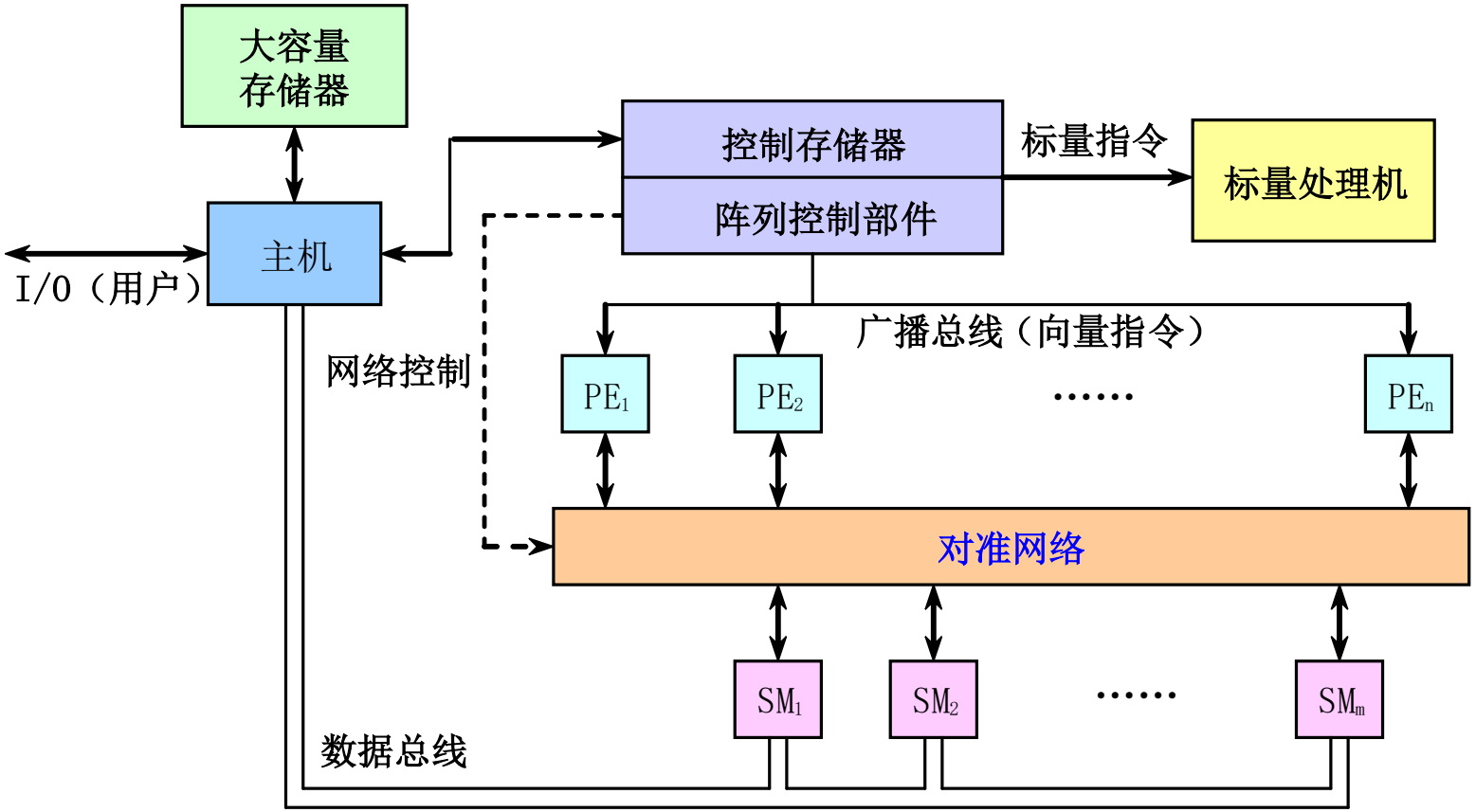
- 集中设置存储器
  - 共享的多体并行存储器SM通过对准网络与各处理单元PE相连。
  - 存储模块的数目等于或略大于处理单元的数目。
- 必须减少存储器访问冲突

  （将数据合理地分配到各存储器模块中 ）
- 在处理单元数目不太多的情况下是很理想的
- 所有阵列指令都必须使用长度为n的向量操作数

  （n为PE的个数)

对准网络(alignment network):实现处理单元和所需访问的存储器对准的互连网络。

13.2 阵列处理机的基本结构



共享存储器的阵列处理机结构

互连网络是共享存储器SM和处理单元PE之间的必由之路。

并行无冲突访问存储器

采用交叉访问方式，一个由m个存储体构成的主存储器，它的速度实际上并不能提高m倍，其根本原因是存在有访问冲突。
产生访问冲突的根源主要有两个，

一是程序中有转移指令；

二是数据的随机性。

后一个问题更为严重，是考虑的重点。

	0 号体	1 号体	2 号体	3 号体
体内地址 0	a_0	a_1	a_2	a_3
1	a_4	a_5	a_6	a_7
2	a_8	a_9	a_{10}	a_{11}
3	…	…	…	…

13.2 阵列处理机的基本结构

一维数组的无冲突访问存储器（一维数组的存储方案）

	0 号体	1 号体	2 号体	3 号体
体内地址 0	a_0	a_1	a_2	a_3
1	a_4	a_5	a_6	a_7
2	a_8	a_9	a_{10}	a_{11}
3	…	…	…	…

如果按连续地址访问，没有冲突。如果按位移量为2的变址方式访问，频带宽度降低一半，即可能有一半地址是冲突的。造成冲突的原因是传统的交叉访问存储器的存储体个数m为2的整数幂，因此变址位移量正好是m的约数。

解决方法：把存储体的个数m选为质数，且m≥向量长度，变址位移量就必然与m互质，一维数组的访问冲突自然也就不存在了。

例如：Burroughs公司巨型机BSP，存储体个数为17
我国研制的银河巨型计算机，存储体的个数为37

13.2 阵列处理机的基本结构

二维数组的无冲突访问存储器

要求：一个n×n的二维数组，按行、列、对角线和反对角线访问，并且在不同的变址位移量情况下，都能实现无冲突访问。

顺序存储：按行、对角线访问没有冲突，但按列访问每次冲突

体内地址	0 号体	1 号体	2 号体	3 号体
0	a_{00}	a_{01}	a_{02}	a_{03}
1	a_{10}	a_{11}	a_{12}	a_{13}
2	a_{20}	a_{21}	a_{22}	a_{23}
3	a_{30}	a_{31}	a_{32}	a_{33}

错位存储：按行、按列访问无冲突，按对角线访问有冲突

体内地址	0 号体	1 号体	2 号体	3 号体
0	a_{00}	a_{01}	a_{02}	a_{03}
1	a_{13}	a_{10}	a_{11}	a_{12}
2	a_{22}	a_{23}	a_{20}	a_{21}
3	a_{31}	a_{32}	a_{33}	a_{30}

二维数组的无冲突访问存储器（方法之一）

将二维数组按列优先或者按行优先的顺序变换为一维数组，以形成一个一维线性地址空间，地址用A表示。

N个PE（AE）和m个存储体的情况。

- 地址映像规则
  - 然后将地址**A**变换成并行存储器地址（**i，j**）。
    其中：**j** 是存储体体号，**j＝A (mod m)**
  - **i**：在相应存储体内的地址，**i=**$\left\lfloor \frac{A}{N} \right\rfloor$ 。
  - 存储体的个数**m**是一个质数。

例： 一个比较简单的例子

设并行存储器的体数m＝7（质数），运算单元数N＝6。

考虑下述4×5的数组：

$$\begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} & a_{04} \\ a_{10} & a_{11} & a_{12} & a_{13} & a_{14} \\ a_{20} & a_{21} & a_{22} & a_{23} & a_{24} \\ a_{30} & a_{31} & a_{32} & a_{33} & a_{34} \end{pmatrix}$$

13.3 阵列处理机实例

按上述地址映像规则，将这个4×5二维数组在m＝7、N＝6的并行存储器中存储的情况：

数组元素	a_{00}	a_{10}	a_{20}	a_{30}	a_{01}	a_{11}	a_{21}	a_{31}	a_{02}	a_{12}	a_{22}	a_{32}	a_{03}	a_{13}	a_{23}	a_{33}	a_{04}	a_{14}	a_{24}	a_{34}
线性地址 A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
体号 j	0	1	2	3	4	5	6	0	1	2	3	4	5	6	0	1	2	3	4	5
体内地址 i	0	0	0	0	0	0	1	1	1	1	1	1	2	2	2	2	2	2	3	3

存储体号 j

体内地址 i	0	1	2	3	4	5	6
0	0 a_{00}	1 a_{10}	2 a_{20}	3 a_{30}	4 a_{01}	5 a_{11}	
1	7 a_{31}	8 a_{02}	9 a_{12}	10 a_{22}	11 a_{32}		6 a_{21}
2	14 a_{23}	15 a_{33}	16 a_{04}	17 a_{14}		12 a_{03}	13 a_{13}
3	21	22	23		18 a_{24}	19 a_{34}	20
4	28	29		24	25	26	27

二维数组的无冲突访问存储器（方法之二）

P. Budnik和D. J. Kuck提出了一种能够实现n×n的二维数组无冲突访问的存储方案。

规则：

（1）并行存储分体数m大于每次要访问的元素个数n，并为质数；

（2）行、列方向上采用不同的错开距离。

符号：δ_1：列上的错开距离。

δ_2：行上的错开距离。

$m=2^{2p}+1$（p为任意自然数）。

无冲突访问的充分条件为：$\delta_1=2p$，$\delta_2=1$。

体内地址	0 号体	1 号体	2 号体	3 号体	4 号体
	a_{00}	a_{01}	a_{02}	a_{03}	
1	a_{13}		a_{10}	a_{11}	a_{12}
2	a_{21}	a_{22}	a_{23}		a_{20}
3		a_{30}	a_{31}	a_{32}	a_{33}

13.2 阵列处理机的基本结构

nxn二维数组中的任意一个元素a_{ij}存放地址为：

体号地址：$y=(i\delta_1+j\delta_2+c) \bmod m$

体内地址：$x=i$

其中，$0\le j\le m-1$，$0\le i\le n-1$，

c为元素a_{00}起始体号地址，m为并行存储体个数。

例：4×4二维数组按行、列、对角线和反对角线访问都不冲突的存储方案为：

体内地址	0 号体	1 号体	2 号体	3 号体	4 号体
0	a_{00}	a_{01}	a_{02}	a_{03}	
1	a_{13}		a_{10}	a_{11}	a_{12}
2	a_{21}	a_{22}	a_{23}		a_{20}
3		a_{30}	a_{31}	a_{32}	a_{33}

主要缺点：浪费存储单元

在n×n二维数组的存储方案中，有(m-n)/m个存储单元浪费。

主要优点：实现简单

列元素按地址顺序存储，行元素按地址取模顺序存储。

13.2 阵列处理机的基本结构

二维数组的无冲突访问存储器（方法之三）

如果不要求同一行中的数组元素按地址顺序存储，则n×n的二维数组实际上只需要用n个并行存储体就能实现按行、列、对角线和反对角线的无冲突访问。这时，并行存储器的利用率最高，没有浪费的存储单元。

规则：对于任意一个n×n的二维数组，如果能够找到满足$n = 2^{2p}$关系的任意自然数p，则这个二维数组就能够使用n个并行存储体实现按行、列、对角线和反对角线的无冲突访问。

4×4数组用4个存储体的无访问冲突存储方案:

	0 号体	1 号体	2 号体	3 号体
体内地址 0	a_{00}	a_{20}	a_{30}	a_{10}
1	a_{21}	a_{01}	a_{11}	a_{31}
2	a_{32}	a_{12}	a_{02}	a_{22}
3	a_{13}	a_{33}	a_{23}	a_{03}

13.2 阵列处理机的基本结构

实现方法：

假设a_{ij}是4×4二维数组中的任意一个元素，其中的下标i和j都可以用两位二进制数表示，假设i和j的高位和低位分别为iH、iL、jH和jL，

a_{ij}的体号地址和体内地址可以通过如下公式计算：

体号地址：2（iL ⊕ jH）＋（iH ⊕ iL ⊕ jL）

体内地址：j 其中，0≤i≤3，0≤j≤3，iH、iL、jH、jL均为0或1。

当数组的维数n大于4时，可以把数组划分成多个4×4的子阵。

例如，可以把一个16×16二维数组划分成4个4×4子阵。

主要优点：没有浪费的存储单元，

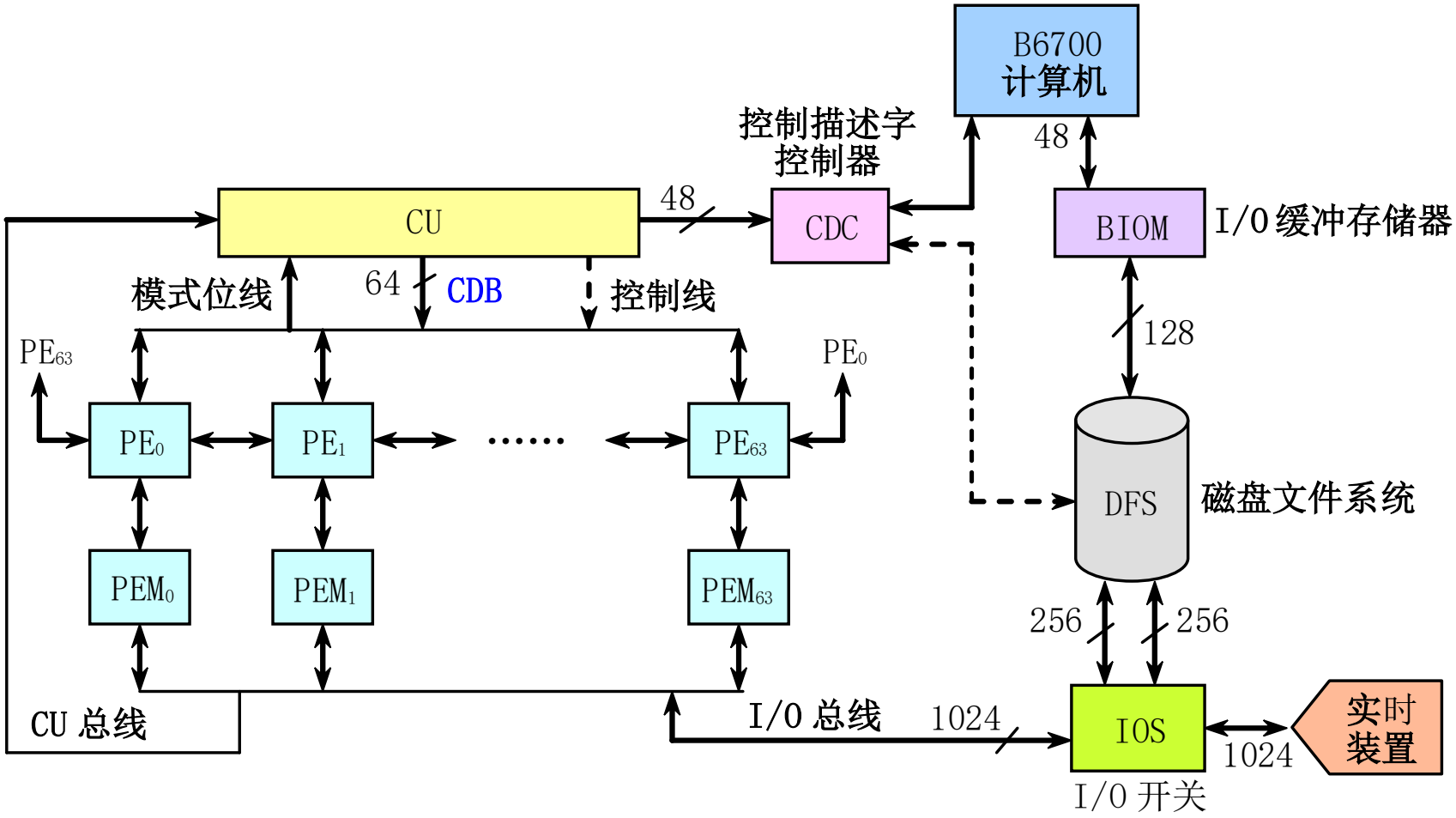
主要缺点：在执行并行读和写操作时需要借助比较复杂的对准网络。

13.3 阵列处理机实例

13.3.1 实例1：Illiac IV阵列处理机

- 美国宝来公司和伊利诺大学合作研制　　1972年
- 最早的阵列处理机
- 一个由3种类型处理机联合组成的多机系统
  - 处理单元阵列：专门用于数组运算
  - 阵列控制器（CU）：既是处理单元阵列的控制部分，又可以看作是一台相对独立的小型标量处理机。
  - 一台标准的B6700计算机：担负Illiac IV输入输出系统和操作系统管理功能

13.3 阵列处理机实例

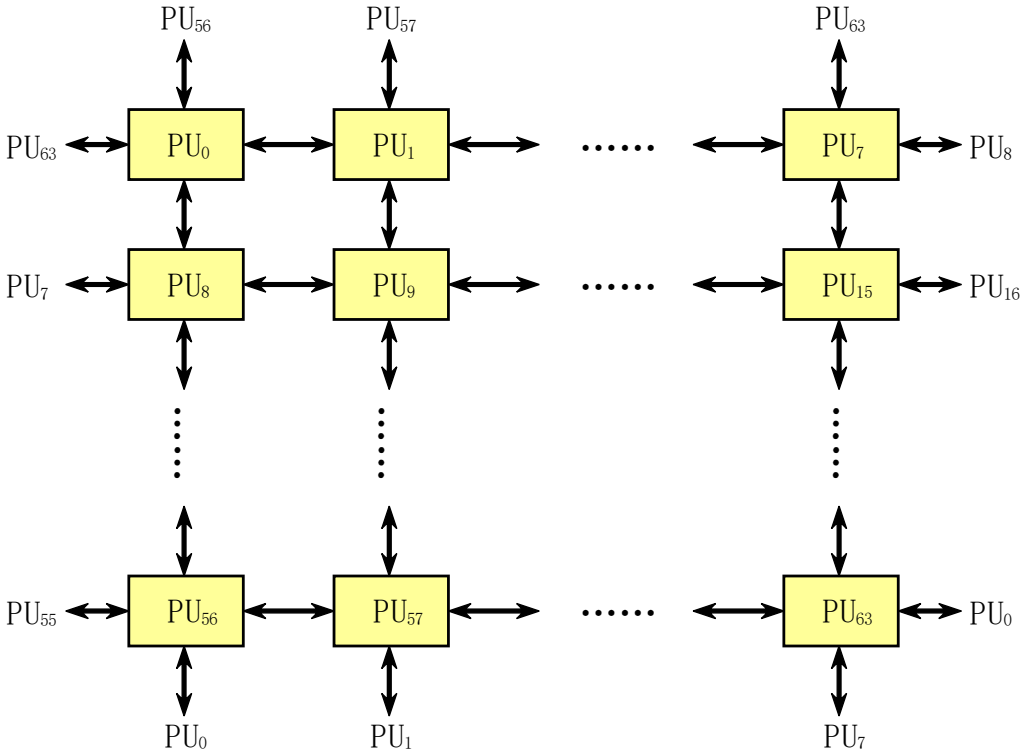


Illiac IV系统总框图

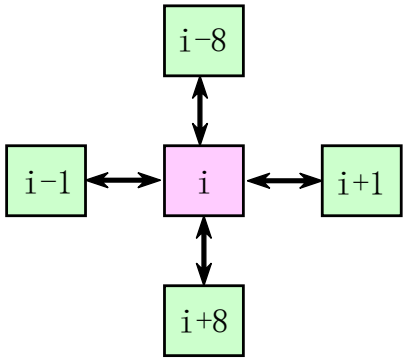
1. Illiac IV阵列

- 由64个处理单元（PE）、64个本地存储器（PEM）和存储器逻辑部件（MLU）组成；
- 把每个PE和PEM对看成是一个处理部件PU；
  - 64个处理部件PU_0～PU_{63}排列成一个8×8方阵
- Illiac IV的阵列结构又称为闭合螺线阵列；
- 既便于一维长向量（多至64个元素）的处理，又便于二维数组运算，以缩短处理单元之间的路径距离。
  - 步距不等于±1或±8的任意处理单元间通信可用软件方法寻找最短路径，其最短距离都不会超过7步。

Illiac IV处理部件的连接



（a）



（b）

例如：从PU_{10}到PU_{46}的距离以下列路径为最短

$$PU_{10} \rightarrow PU_9 \rightarrow PU_8 \rightarrow PU_0 \rightarrow PU_{63} \rightarrow PU_{62} \rightarrow PU_{54} \rightarrow PU_{46}$$

- 一般情况，n×n个单元组成的阵列中，任意两个处理单元之间的最短距离不会超过（n－1）步。

➢ 每个处理单元有6个可编程序寄存器

- 64位字长的累加器RGA
- 64位字长的操作数寄存器RGB
- 64位字长的数据路由寄存器RGR
- 64位字长的通用寄存器RGS

  （可被程序用来暂存中间结果）
- 16位的变址寄存器
- 8位的模式寄存器

  （存放PE屏蔽信息以及状态位）

13.3 阵列处理机实例

- 运算部件
  - 加/乘算术单元
  - 逻辑单元
  - 移位单元
  - 地址加法器等
- 操作数来源
  - PE本身的寄存器
  - PEM
  - CU的公共数据总线
  - PE的4个近邻

- 并行的加法速度

  每秒1010次8位定点加法或150×106次64位浮点加法

- 每一个处理单元有一个自己的本地存储器PEM
- PE和PEM之间经过存储器逻辑部件MLU相连

2. 阵列控制器CU

- 一台小型计算机
  - 对阵列的处理单元进行控制
  - 利用本身的内部资源执行一整套指令，用以完成标量操作。

- 功能
  - 对指令流进行控制和译码，包括执行一整套标量指令；
  - 向各处理单元发出执行数组操作指令所需的控制信号；
  - 产生并向所有处理单元广播公共的地址部分；
  - 产生并向所有处理单元广播公共的数据；
  - 接收和处理由各**PE**计算出错、系统**I/O**操作以及**B6700**所产生的陷阱中断信号。
- 阵列控制器CU与处理单元之间有4条信息通路
  - **CU**总线
  - 公共数据总线**CDB**
  - 模式位线
  - 指令控制线（大约有**200**根）

3. 输入输出系统

由磁盘文件系统DFS、I/O分系统和B6700管理计算机组成。

- 磁盘文件系统DFS
  - 两套大容量并行读写磁盘系统及其相应的控制器；
  - 每套有13台磁盘机，总容量为109位；
  - 每台磁盘机有128道，每道一个磁头，并行读写，数据宽度为256位，最大传输率为502×106b/s；平均等待时间为19.6ms；
  - 如果两个通道同时发送或接收数据，则数据宽度为512位，最大传输率为109b/s。

- I/O系统

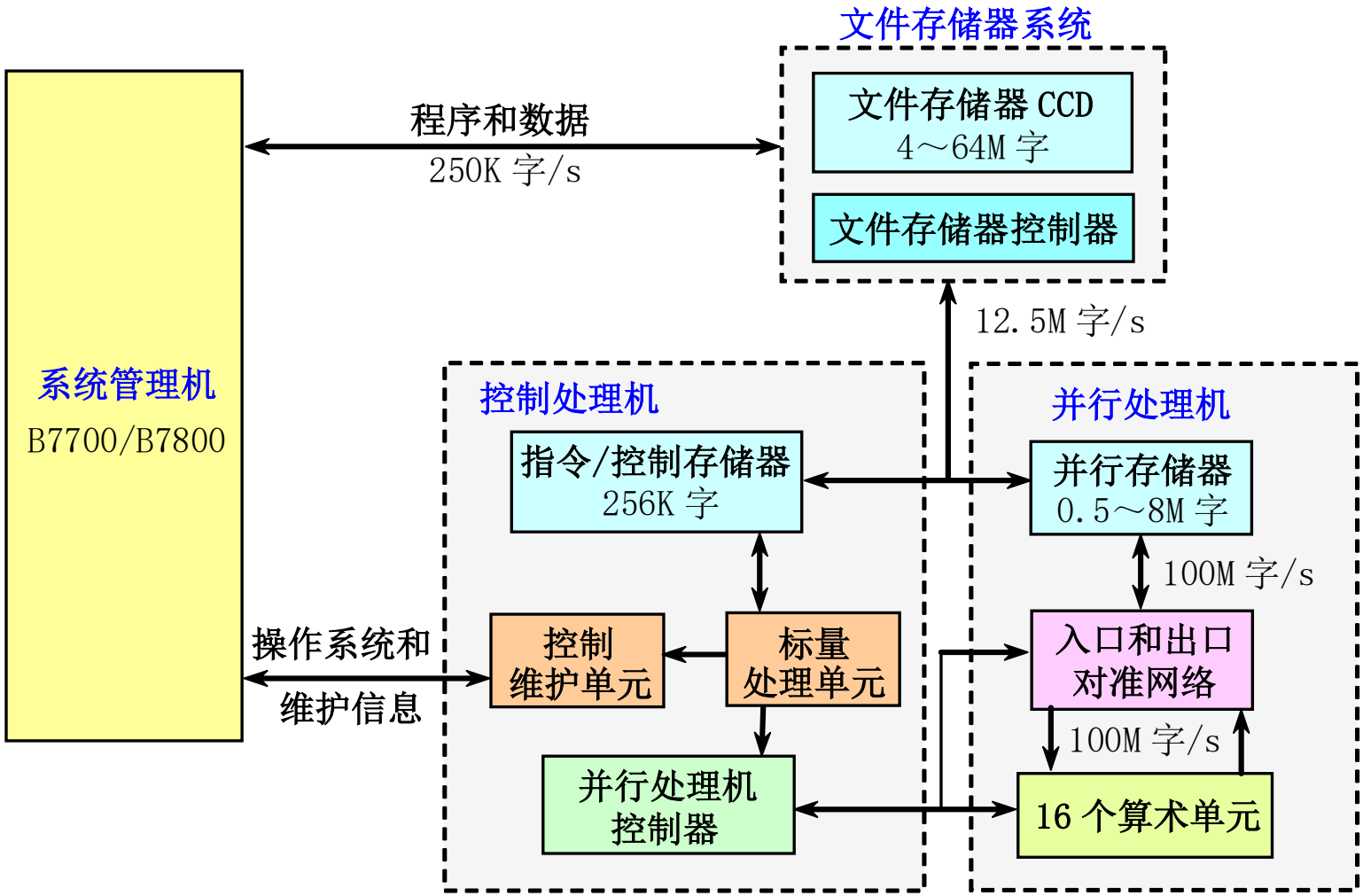
包括3部分：

- 输入/输出开关IOS
  - 作为一个开关，把DFS或可能连上的实时装置转接到阵列存储器，进行大批数据的I/O传送；
  - 作为DFS和PEM之间的缓冲，以平衡两边不同的数据宽度。
- 控制描述字控制器CDC
  - 对阵列控制器CU的I/O请求进行管理
- **BIOM**
  - 在**DFS**和**B6700**之间，是为了取得二者之间传送带宽上的匹配。

- B6700管理计算机
  - 管理全部系统资源，完成用户程序的编译或汇编，
  - 为Illiac IV进行作业调度、存储分配、产生I/O控制描述字送至CDC、处理中断、提供操作系统所具备的其他服务等。

13.3.2 实例2：BSP计算机

- 美国宝来公司和伊利诺依大学 1979年
- 共享存储器结构的SIMD计算机的典型代表
- 最高处理性能：每秒5千万次浮点运算
- 依靠并行性来提高性能
- BSP处理机由3部分构成：控制处理机，并行处理机，文件存储器。

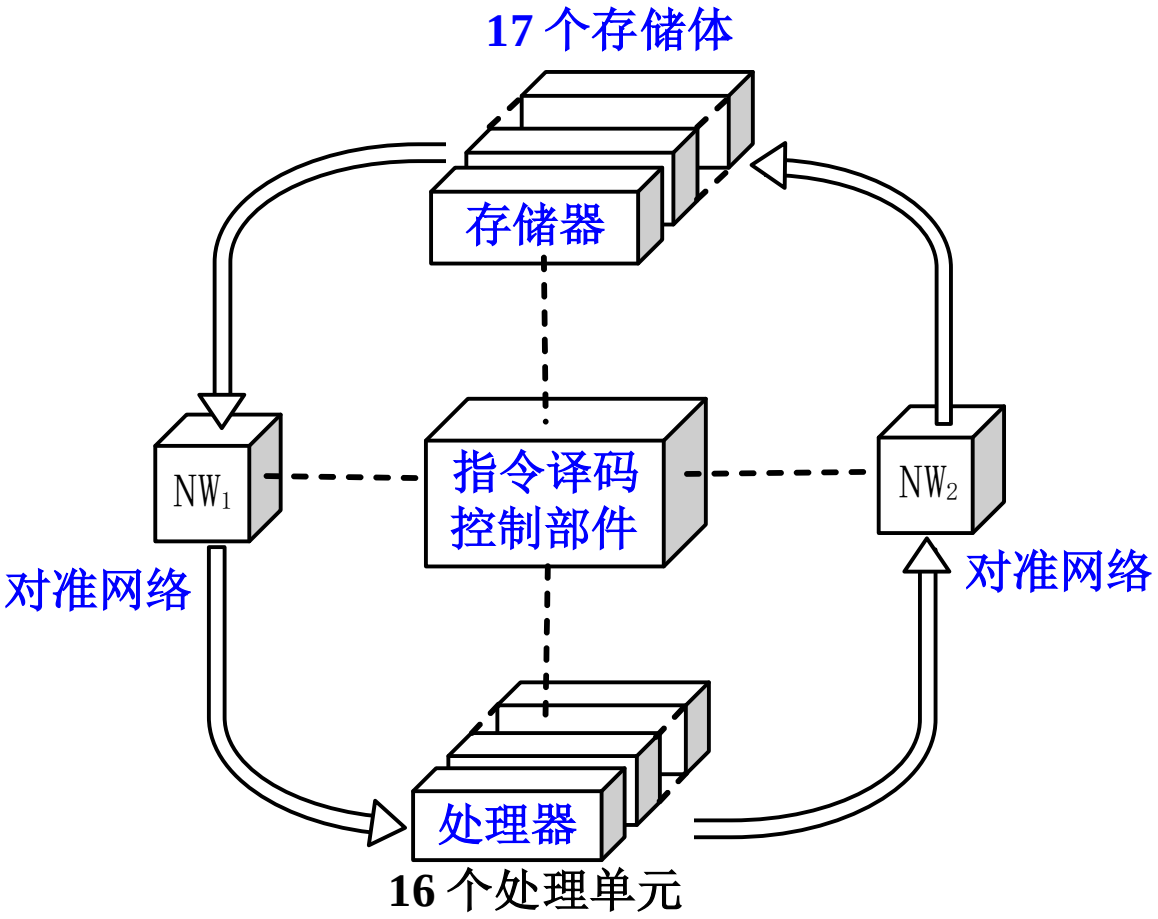


BSP计算机系统的框图

1. BSP处理机

- 并行处理机
  - 包含16个算术单元AE、由17个存储体组成的一个无冲突访问的并行存储器和两套对准网络（分别为入口和出口对准网络）
  - 一条5级的数据流水线
    - 从17个存储器输出端口并行读出16个操作数；
    - 经对准网络NW_1将16个操作数重新排列，形成16个算术单元所需要的顺序；
    - 将排列好的16个操作数送到16个算术单元进行处理；
    - 所得的16个结果经对准网络NW_2重新排列成在17个存储体中存储所需要的次序；
    - 写入并行存储器。

13.3 阵列处理机实例



BSP的5级数据流水线结构示意图

13.3 阵列处理机实例

- 两套对准网络的作用：在读或写并行存储器时，使并行存储器中为保证无冲突访问而错开存放的操作数顺序能够与算术单元并行处理所要求的正常顺序协调一致。
- 这种流水线对提高系统处理效率有很大作用 。
  - 有效地实现了处理单元、存储器和互连网络在时间上重叠工作，在理想情况下能取得带宽的完全匹配。
  - 可把大于16的任意长度的向量按16个分量的标准长度分为若干段，依次在时间上重叠起来进行处理。
  - 实现不同向量指令的重叠执行。
- 数据保存在由17个存储体组成的并行存储器中，每个存储体的容量可达512K字，存储周期为160ns。

  （一个无冲突访问存储器 ）

➢ 控制处理机

控制并行处理机，提供与系统管理机相连的接口。

- 标量处理单元：处理存储在指令/控制存储器中的全部操作系统和用户程序的指令。
- 全部的向量指令以及某些成组运算的标量指令被送给并行处理机控制器。在经过合格性检查之后，并行处理机控制器将指令转换为微操作序列去控制16个AE操作。
- 指令/控制存储器的容量为256K字，存储周期为160ns，字长为56位，其中8位是校验位，提供单错校正和双错检测的能力。
- 控制维护单元：系统管理机与控制处理机的接口，用来对控制处理机进行初始化以及监控命令的通信和维护。

- 文件存储器
  - BSP直接控制下的唯一外围设备。
  - BSP程序执行过程中所产生的暂存文件和输出文件都是先存放在文件存储器中，然后才被送给系统管理机，输出给用户。
  - 文件存储器的数据传输率较高，大大缓解了I/O受限问题 。

2. BSP并行存储器

- 由17个存储体组成
- 可以实现无冲突访问

13.3 阵列处理机实例

实现无冲突访问的硬件支持：

- 质数个存储器端口（存储体数是质数17）
- 存储端口和AE之间的交叉开关（对准网络）
- 特殊的存储器地址生成机构

13.4 阵列处理机的并行算法举例

以Illiac IV为例，讨论阵列处理机的算法。

1. 有限差分问题

- 把一个有规则的网格覆盖在整个场域上，用网格点上的变量值写出差分方程组以代替场方程来进行计算。
- 描述平面场的拉普拉斯方程(Laplace's equation)

$$\frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} = 0$$

Pierre-Simon Laplace(1749 - 1827)

- 将二阶偏导数表示为差分形式

$$\frac{\partial^2 U}{\partial x^2}=\frac{U(x+h,y)-2U(x,y)+U(x-h,y)}{h^2}$$

$$\frac{\partial^2 U}{\partial y^2}=\frac{U(x,y+h)-2U(x,y)+U(x,y-h)}{h^2}$$

- 代入原方程，则可得有限差分计算公式

$$U(x,y)=\frac{U(x+h,y)+U(x,y+h)+U(x-h,y)+U(x,y-h)}{4}$$

（*x*，*y*）：平面网格点坐标　　*h*：网格间距

任一网格点(x, y)上的函数值可由其四周邻近点的函数值计算出来，正好反映了阵列处理机每一处理单元与其四个近邻连接的性质。

网格边缘的函数值是已知的，由场域的边界条件决定；而对于内部各点的函数值，开始时可选择为零，多次迭代直至连续二次迭代所求值的差小于规定误差为止(已知迭代过程是收敛的)。

- 差分法求解的精度与网格间距有直接的关系，网格越小，精度越高，但求解所花费的时空开销越大。
- Illiac Ⅳ在计算时，是把内部网格点分配给各个处理单元的。因此，上述计算过程可以并行地完成，从而大幅度地提高处理速度。

2. 矩阵加

考虑两个8×8的矩阵A和B的相加，所得结果矩阵C也是一个8×8的矩阵。

- 把A、B、C中位于相应位置的分量存放在同一PEM内。

假设：

- A的分量在全部64个PEM中存放的单元地址都是α；
- B的全部分量的地址都是α＋1；
- C的全部分量的地址都是α＋2。

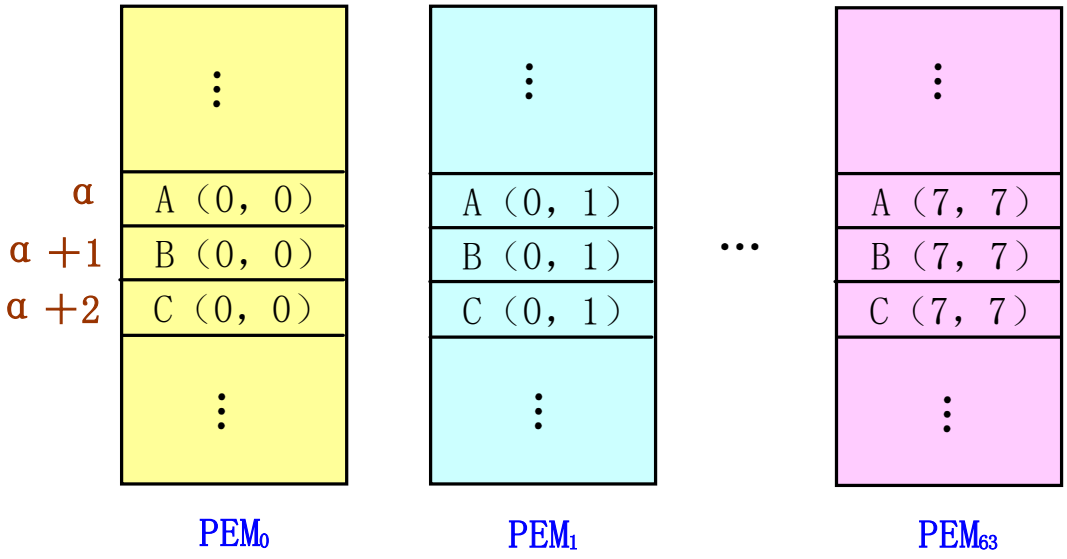
- 用3条Illiac IV的汇编指令就可以实现矩阵相加。

13.4 阵列处理机的并行算法举例

LDA　　ALPHA　　；全部A的分量由PEM_i送PE_i的累加器RGA_i

ADRN　　ALPHA＋1　　；全部B的分量与（RGA_i）进行浮点加，结果送RGA_i

STA　　ALPHA＋2　　；全部（RGA_i）由PE_i送PEM_i的α＋2单元



矩阵相加存储器分配举例

3. 矩阵乘

设A、B和C为3个8×8的二维矩阵。若给定A和B，则C＝A*B的64个分量可利用下列公式计算。

$$c_{ij} = \sum_{k=0}^{7} a_{ik} b_{kj} \qquad 0 \leqslant \mathrm{i}，\ \mathrm{j} \leqslant 7$$

- 在SISD计算机上求解，执行下列FORTRAN程序：

```
DO 10 I＝0，7
DO 10 J＝0，7
C（I，J）＝0
DO 10 K＝0，7
10  C(I，J)＝C(I，J)＋A(I，K)*B(K，J)
```

三重循环，每重循环执行8次，共需512次乘加的时间。

在SIMD阵列处理机上求解这个问题

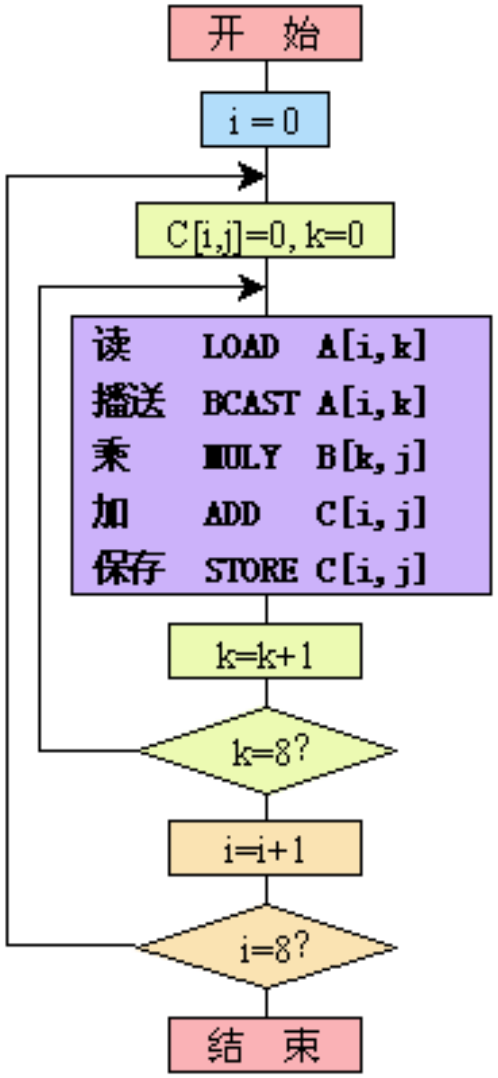
执行下列FORTRAN程序：

```
DO 10 I＝0，7
C（I，J）＝0
DO 10 K＝0，7
10  C(I，J)＝C(I，J)＋A(I，K)*B(K，J)
```

速度提高到原来的8倍，即每个处理单元的计算时间缩短为64次乘加时间。

13.4 阵列处理机的并行算法举例

程序流程图：



RGA(J) = A(I,J)
RGA(J) = RGA(K)
RGA(J) = RGA(J)*B (K,J)
RGA(J) = RGA(J)+C (I,J)
C(I,J) = RGA(J)

13.4 阵列处理机的并行算法举例

在3*3矩阵相乘中理解：

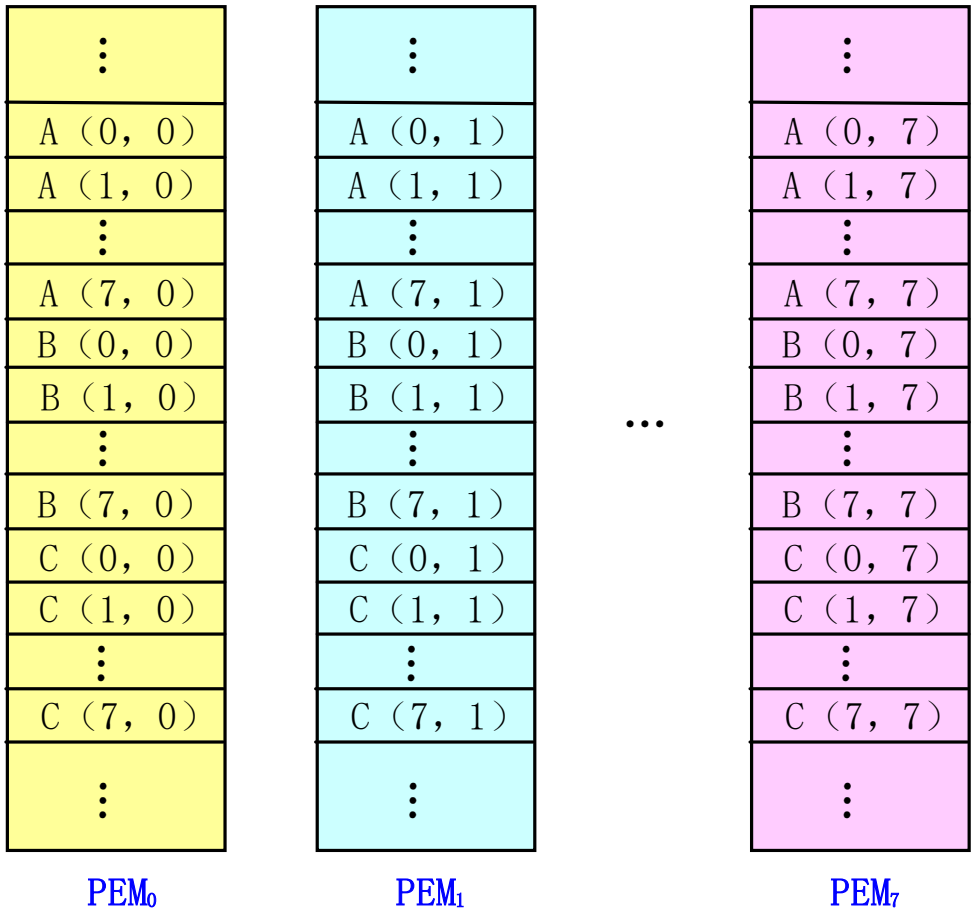
$$\begin{pmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \\ b_{20} & b_{21} & b_{22} \end{pmatrix} =$$

$$\begin{pmatrix} a_{00}b_{00}+a_{01}b_{10}+a_{02}b_{20} & a_{00}b_{01}+a_{01}b_{11}+a_{02}b_{21} & a_{00}b_{02}+a_{01}b_{12}+a_{02}b_{22} \\ a_{10}b_{00}+a_{11}b_{10}+a_{12}b_{20} & a_{10}b_{01}+a_{11}b_{11}+a_{12}b_{21} & a_{10}b_{02}+a_{11}b_{12}+a_{12}b_{22} \\ a_{20}b_{00}+a_{21}b_{10}+a_{22}b_{20} & a_{20}b_{01}+a_{21}b_{11}+a_{22}b_{21} & a_{20}b_{02}+a_{21}b_{12}+a_{22}b_{22} \end{pmatrix}$$

PEM0　　PEM1　　PEM2

13.4 阵列处理机的并行算法举例

- A、B、C向量在处理部件存储器中的存放



4. 累加和

一个将N个数的顺序相加转变为并行相加的问题。

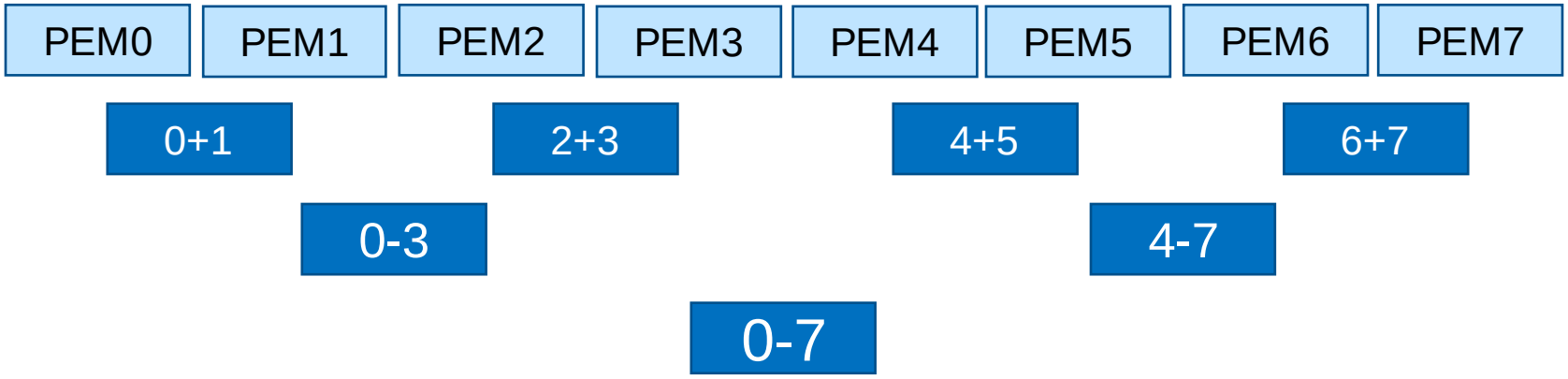
- 只有处于活动状态的处理单元才能执行相应的操作。
- 取N＝8。即有8个数A(I)要顺序累加（0≤I≤7）
- 在SISD计算机上可写成下列FORTRAN程序：

```
C＝0
DO 10 I＝0，7
10  C＝C＋A（I）
```

这是一个串行程序，共要进行8次加法。

13.4 阵列处理机的并行算法举例

- 在阵列处理机上采用成对递归相加的算法，则只需$\log_2 8=3$次加法。



一般性算法：首先，把原始数据A（I），$0\le I\le 7$，分别存放到8个PEM的α单元中，然后按照下面的步骤求累加和：

13.4 阵列处理机的并行算法举例

Step 1 置全部PE_i为活动状态，$0 \le i \le 7$；

Step 2 全部A(I) , $0 \le I \le 7$,

从PEM_i的α单元读到相应PEi的累加寄存器RGA_i中，$0 \le i \le 7$；

Step 3 令K＝0；

Step 4 将全部PE_i的（RGA_i）传送到RGR_i，$0 \le i \le 7$；

Step 5 全部PE_i的（RGR_i）经过互连网络向右传送2^K步距，$0 \le i \le 7$；

Step 6 $j = 2^K - 1$；

Step 7 置PE_0至PE_j 为不活动状态；

Step 8 处于活动状态的所有PE_i执行：$(RGA_i)=(RGA_i)+(RGR_i)$　$j < i \le 7$；

Step 9 K＝K+1 ；

Step 10 若K＜3，则转回第四步，否则继续往下执行；

Step 11 置全部PE_i为活动状态，$0 \le i \le 7$；

Step 12 全部PE_i的（RGA_i）存入相应的PEM_i的α＋1单元中，$0 \le i \le 7$。

13.4 阵列处理机的并行算法举例

计算过程示意图：