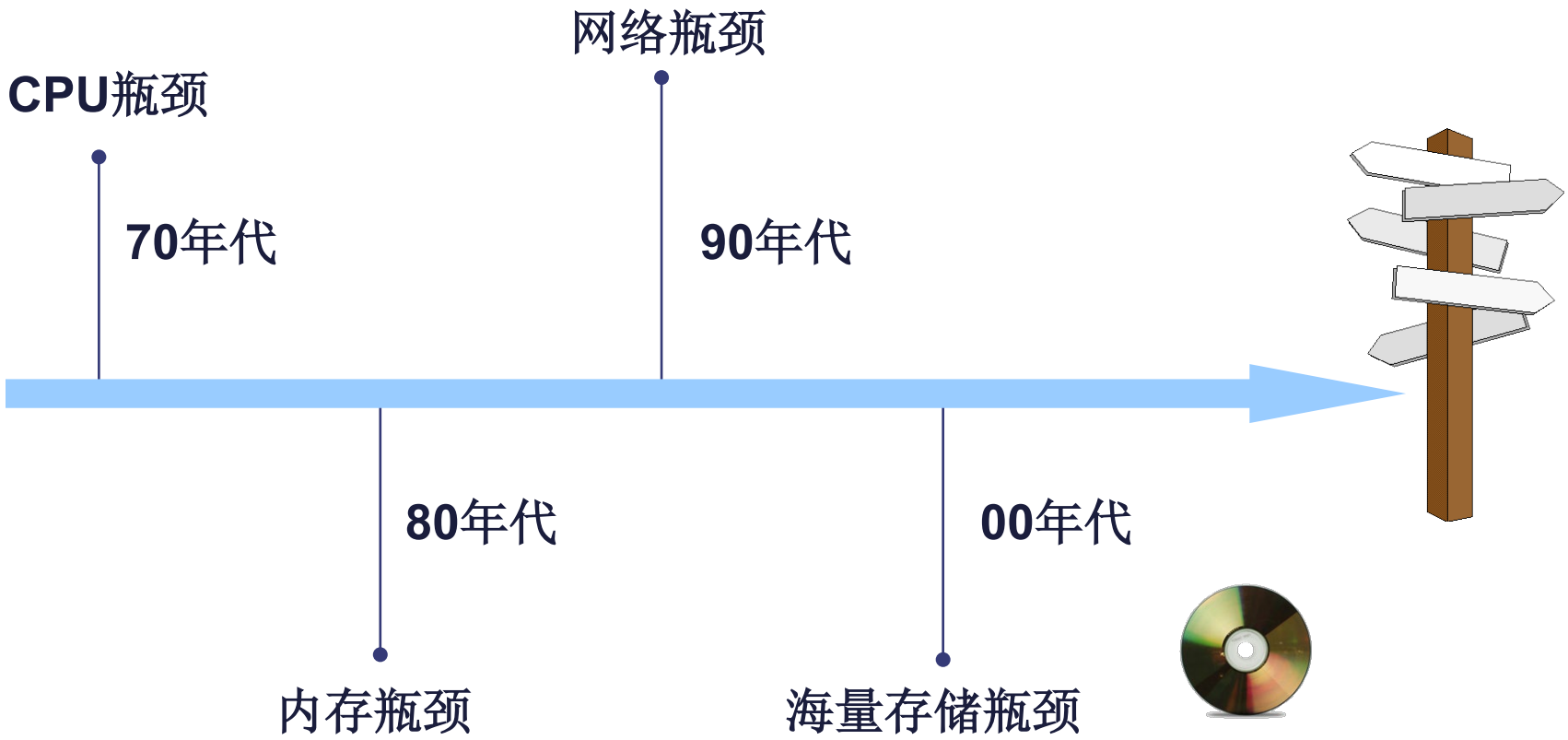
计算机系统结构

Computer Architecture

华中科技大学教育部信息存储系统重点实验室

武汉光电国家研究中心信息存储研究部

性能：瓶颈的转移 应用的转移



共享处理机

共享存储器数据

第一章
计算机体系结构的基础知识

- 引言
- 计算机体系结构的概念/分类
- 定量分析技术基础
- 计算机体系结构的发展

教学要求：

1. 熟练掌握内容：

计算机系统层次结构；系统结构、计算机组成和计算机实现定义，三者关系；透明性；

Amdahl定律，CPU性能公式，CPI，局部性原理，MIPS定义。

2. 掌握内容：

系统结构分类，冯·诺依曼计算机特征。

3. 了解内容：

计算机系统结构的发展，计算机体系结构中并行性的发展。

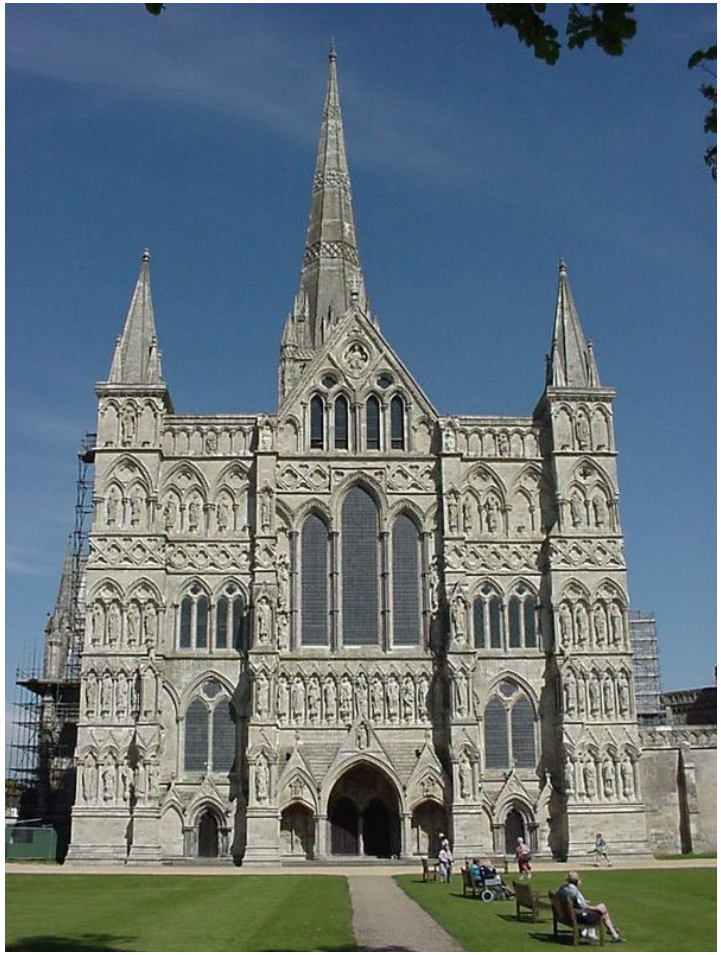
引 言

1. 计算机性能的高速增长受益于：
   - 电路技术的发展
   - 体系结构技术的发展
2. 体系结构的重要性
   - 80年代 RISC,指令级并行
   - 2004Intel 转向多核，线程级数据级并行

1.1　计算机体系结构的基本概念

英文名称：Computer Architecture

Architecture的英文原义是“建筑学”



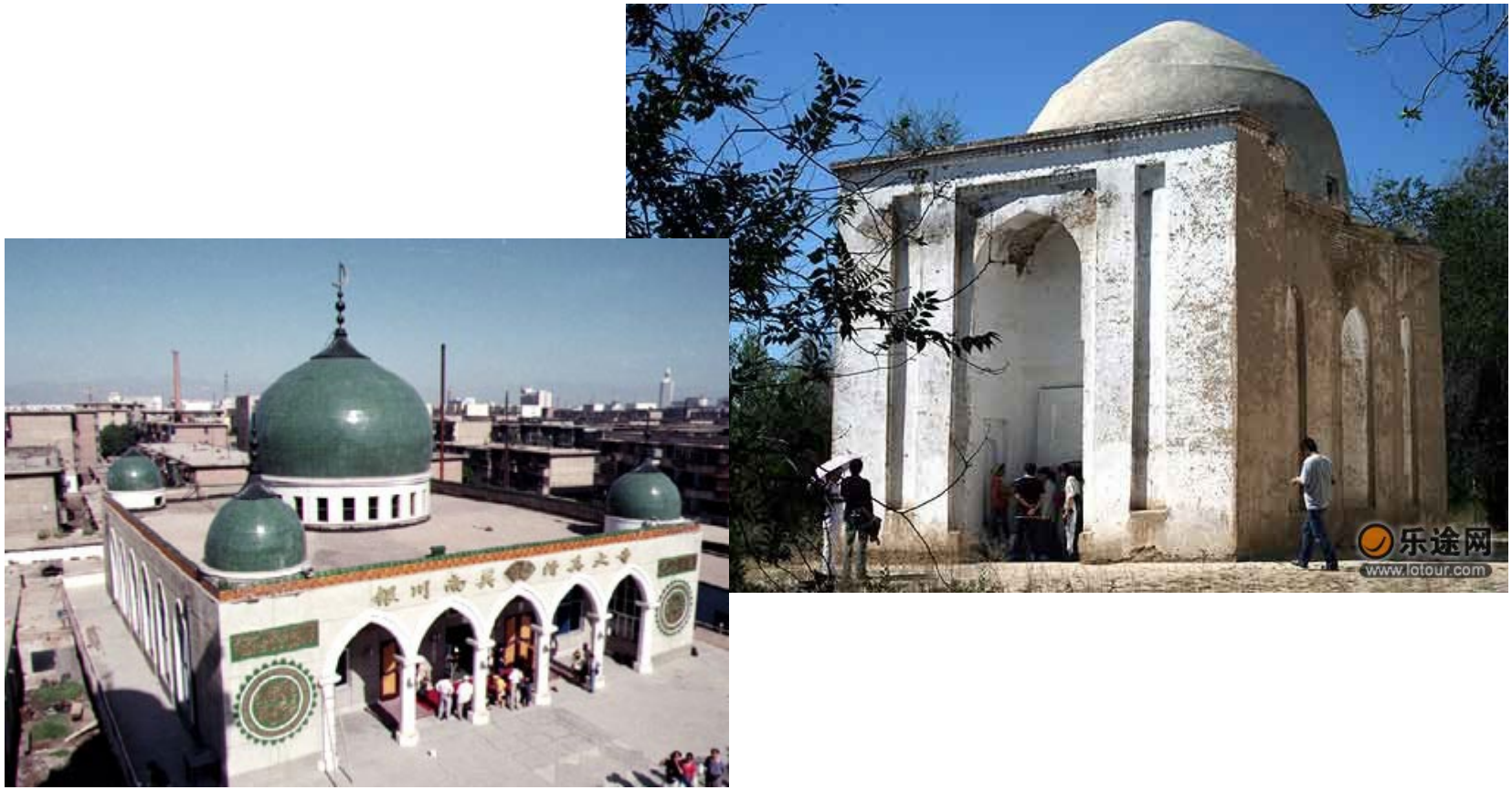
法国沙特尔主教堂

米兰大教堂

哥特式建筑：尖塔高耸、尖形拱门、大窗户及绘有圣经故事的花窗玻璃。

利用尖肋拱顶、飞扶壁、修长束柱，营造出轻盈修长的飞天感。以及新的框架结构以增加支撑顶部的力量，使整个建筑以直升线条、雄伟的外观和教堂内空阔空间，再结合镶着彩色玻璃的长窗，使教堂内产生浓厚宗教气氛。

1.1　计算机体系结构的基本概念



“计算机系统结构”作为事物名称：使用者必须了解的机器外部特性知识（广义定义）。在本课程中“使用者”目前特指最低级语言程序员，“外部特性”特指整个硬件的外部特性（狭义定义）。

计算机系统结构（也叫“计算机体系结构”）：传授计算机整机 （硬软件统一条件下）设计的重大技术知识。

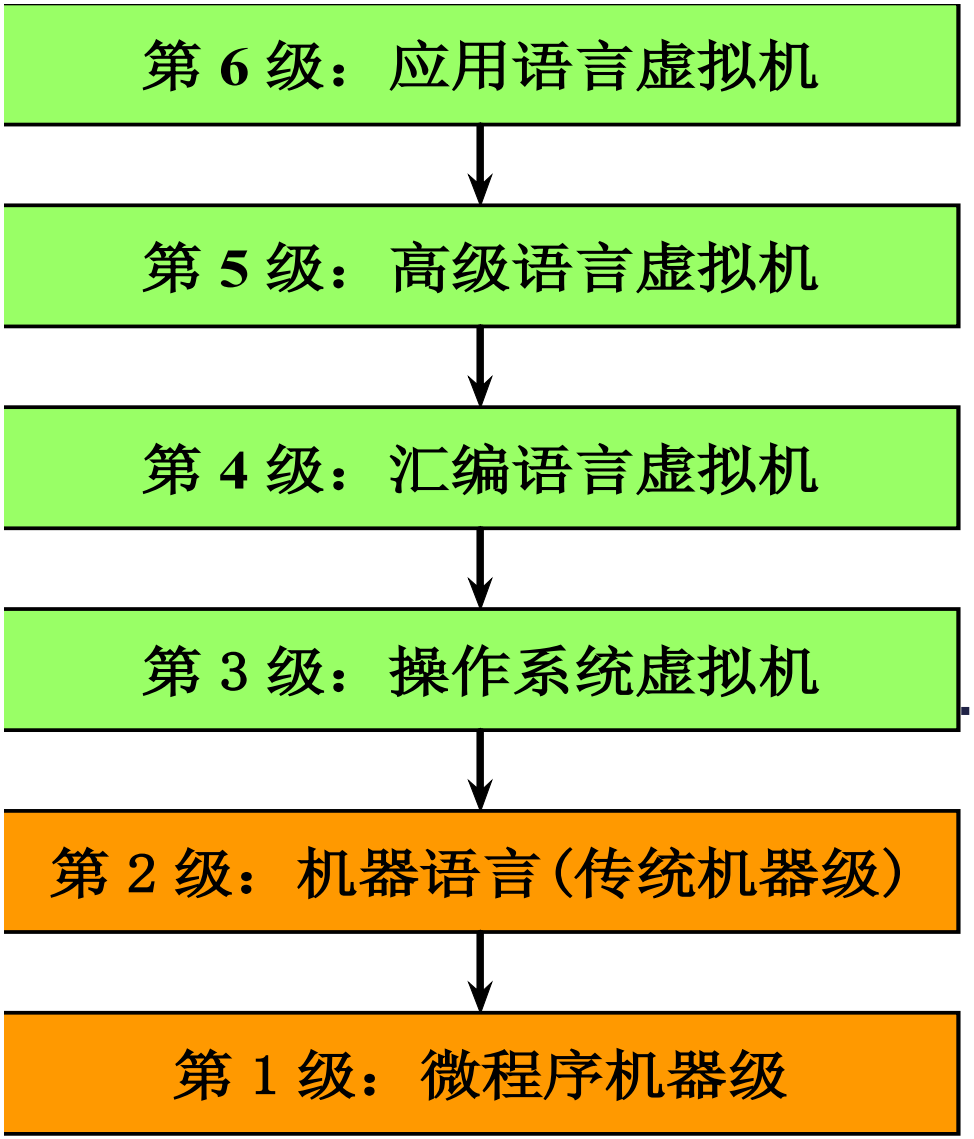
不同的时期对计算机系统结构有着不同的定义

1.1.1 计算机系统中的层次概念

1. 计算机系统＝软件＋硬件/固件
2. 计算机语言由低级向高级发展

　　高一级语言的语句相对于低级语言功能更强，更便于应用，但又都以低级语言为基础。

3. 从计算机语言的角度，把计算机系统按功能划分成多级层次结构。


第 6 级：应用语言虚拟机
第 5 级：高级语言虚拟机
第 4 级：汇编语言虚拟机
第 3 级：操作系统虚拟机
第 2 级：机器语言(传统机器级)
第 1 级：微程序机器级
软件
硬件或固件

1.1.2　计算机体系结构定义

1. 计算机体系结构定义：

程序员所看到的计算机属性，即概念性结构与功能特性。

2. 按照计算机系统的多级层次结构，不同级程序员所看到的计算机具有不同属性。

3. 透明性

在计算机技术中，对这种本来是存在的事物或属性，但从某种角度看又好象不存在。

4. Amdahl提出的体系结构：**传统机器级体系结构**。

即一般所说的机器语言程序员所看到的传统机器级所具有的属性。

5. 对于通用寄存器型机器，这些属性主要指：

(1) **数据表示**　（硬件能直接辩认和处理的数据类型）

(2) **寻址规则**　（最小寻址单元、寻址方式及其表示）

(3) **寄存器定义**　（各种寄存器的定义、数量和使用方式）

(4) **指令集**　（机器指令的操作类型和格式、指令间的排序和控制机构等）

(5) **中断系统**　（中断类型和中断响应硬件的功能等）

(6) 机器工作状态的定义和切换　（管态和目态等）

(7) 存储系统　（主存容量、程序员可用的最大存储容量等）

(8) 信息保护　（信息保护方式和硬件对信息保护的支持）

(9) I/O结构　（I/O连接方式、处理机/存储器与I/O设备间数据传送的方式和格式以及I/O操作的状态等）

经典计算机体系结构概念的实质：

计算机系统中软硬件界面的确定，界面之上是软件功能，界面之下是硬件和固件功能。

1.1.3 计算机组成和计算机实现技术

1. **计算机组成**：计算机体系结构的逻辑实现。
2. **计算机实现**：计算机组成的物理实现。

   一种体系结构可以有多种组成。

   一种组成可以有多种物理实现。

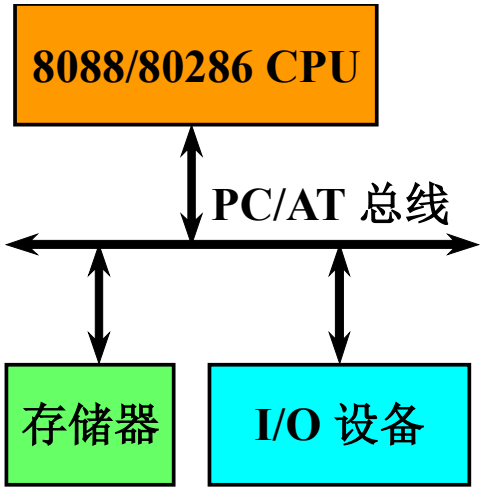
3. **系列机/兼容机/软件兼容**

   (1) 系列机：在一个厂家内生产的具有相同体系结构，但具有不同组成和实现的一系列不同型号的机器。

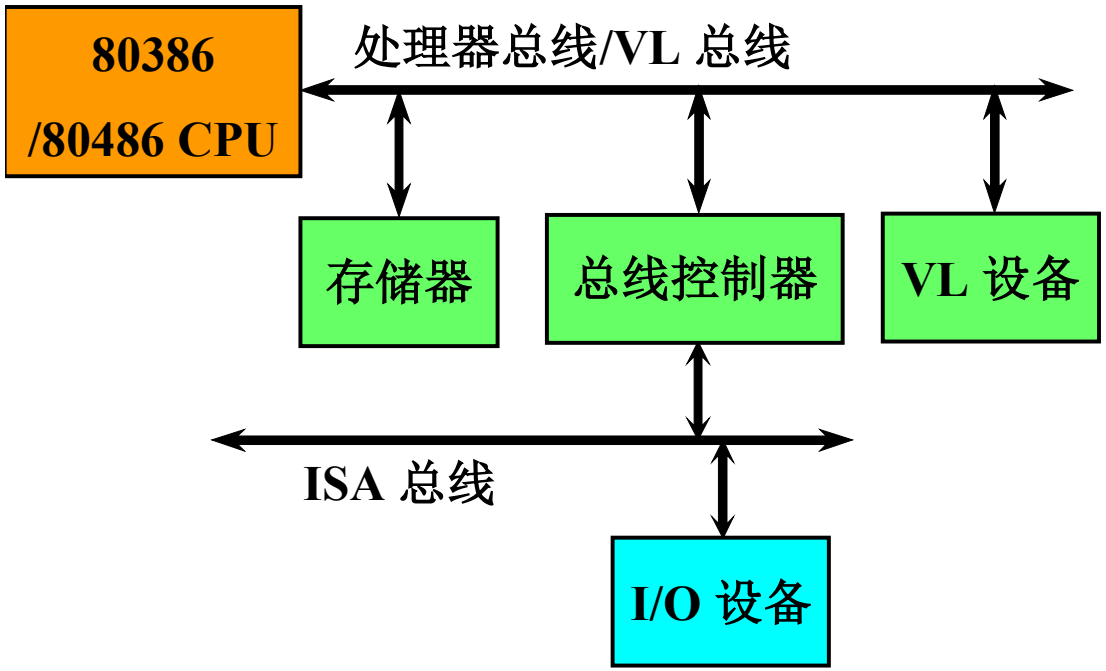
   如：IBM 370系列有370/115、125、135、145、158、168等一系列从低速到高速的各种型号。

(2) IBM PC系列机

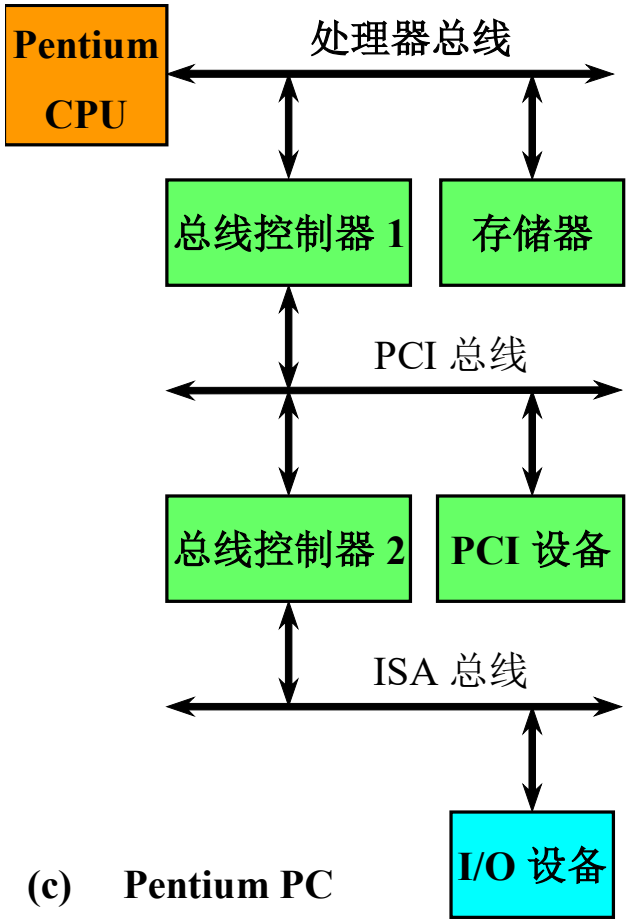
（处理器、处理器字宽、主要I/O总线、存储空间、主要操作系统和计算机结构）



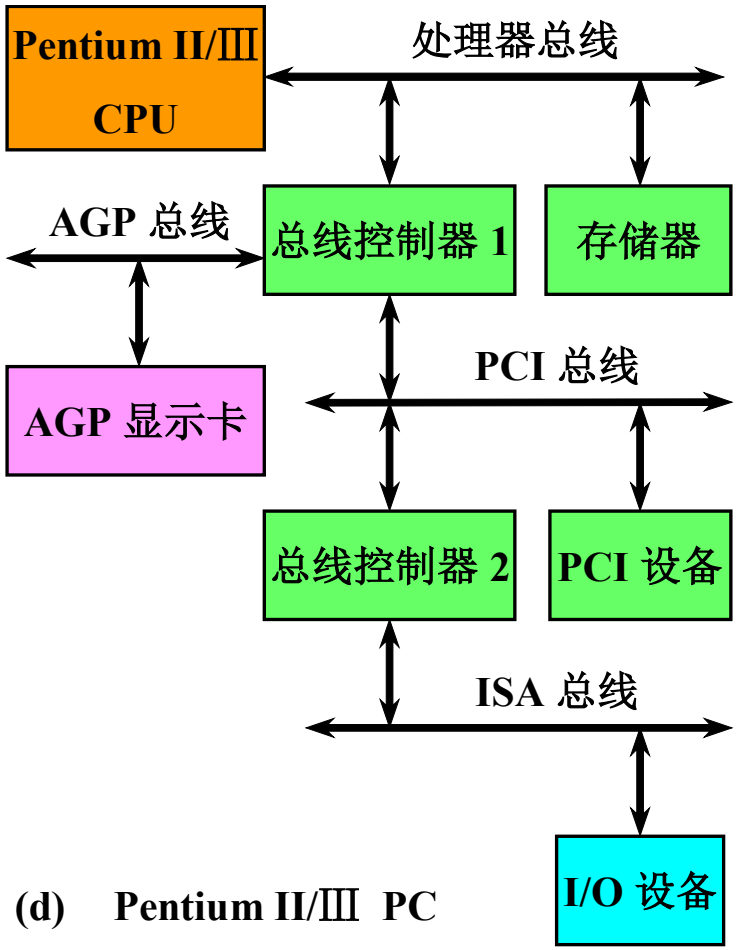
(a)　PC、XT 和 PC AT



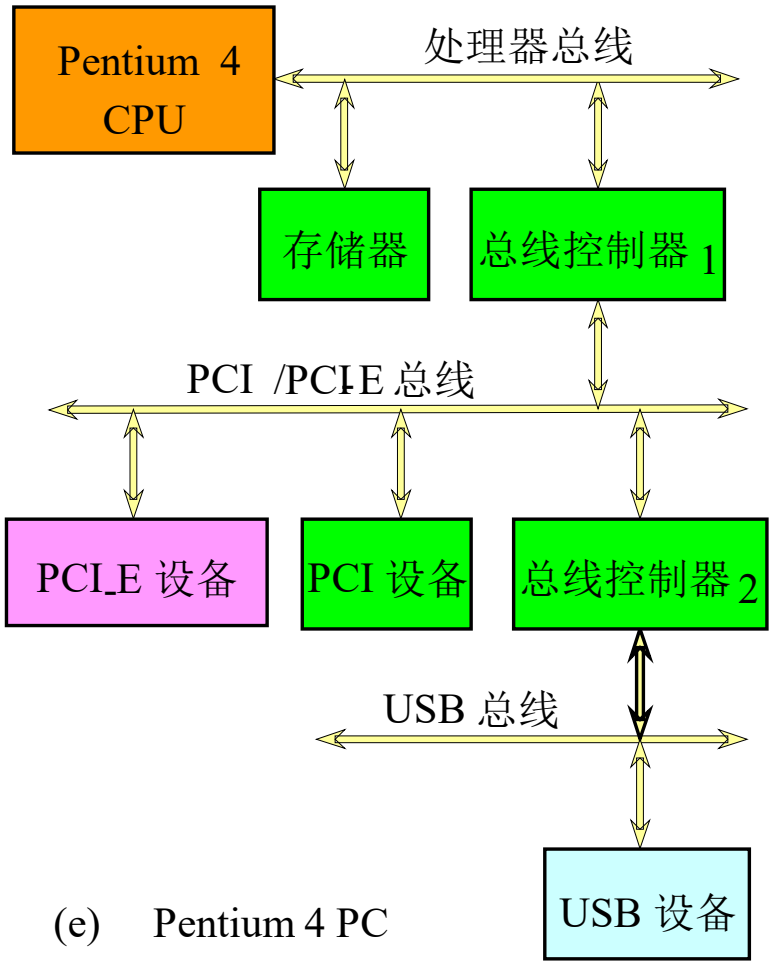
(b)　80386/80486 PC



(c) Pentium PC



(d) Pentium II/III PC



(e) Pentium 4 PC

1.1.4 计算机的分代和分类

1. 计算机到目前为止已经发展了五代

这五代计算机分别具有明显的器件、体系结构技术和软件技术的特征。

2. 计算机可以根据价格分为五个档次：

巨型机、大型机、中型机、小型机、微型机

佛林（Flynn）分类法：1966年Michael J Flynn提出，按照指令流和数据流的多倍性特征对计算机系统进行分类。

指令流：机器执行的指令序列

数据流：由指令流调用的数据序列，包括输入数据和中间结果

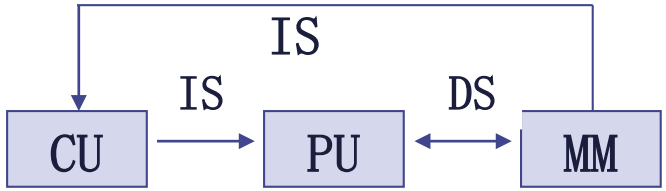
多倍性（multiplicity）：在系统性能瓶颈部件上同时处于同一执行阶段的指令或数据的最大可能个数

1. 单指令流单数据流（SISD---Single Instruction Stream Single Data Stream）
2. 单指令流多数据流（SIMD---Single Instruction Stream Multiple Data Stream）
3. 多指令流单数据流（MISD---Multiple Instruction Stream Single Data Stream）
4. 多指令流多数据流（MIMD---Multiple Instruction Stream Multiple Data Stream）

单指令流单数据流SISD (Single Instruction Single Datastream)：

传统顺序处理机

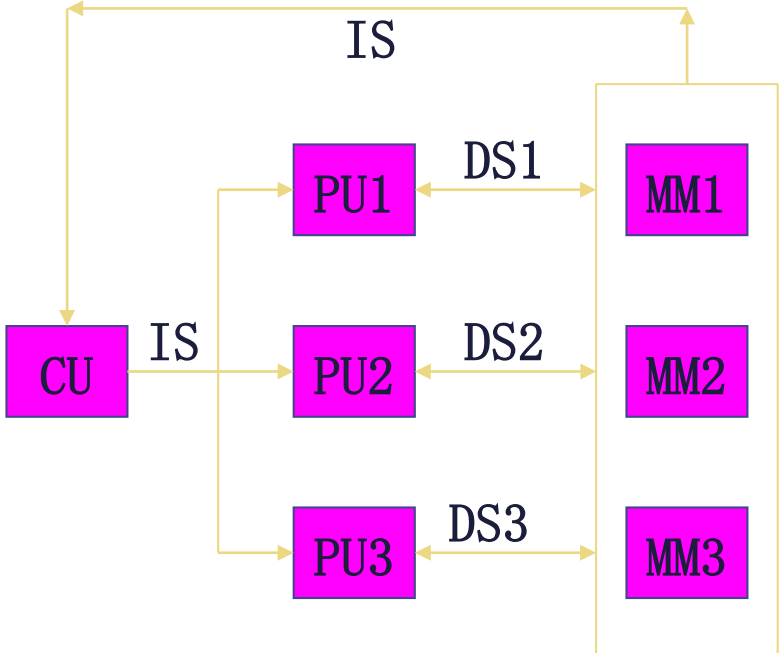
SISD：包括单功能部件处理机，多功能部件处理机，流水线处理机 —— 标量流水线处理机



CU：控制部件
PU：处理部件
MM：存储器模块
IS：指令流
DS：数据流

单指令流多数据流SIMD(Single Instruction Multiple Datastream)：

SIMD：包括并行处理机，阵列处理机，向量处理机，超标量处理机，超流水线处理机，即多个PU按一定的方式互连，在同一个CU控制下，并行的对多个数据进行处理。

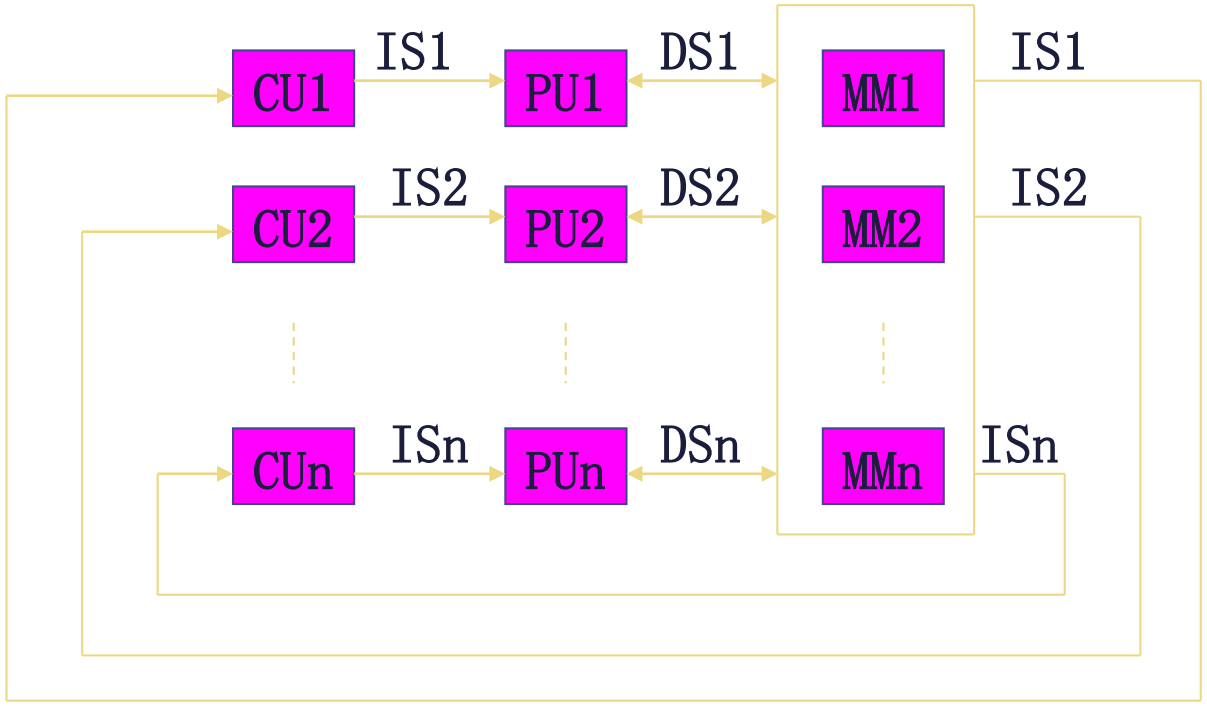


从CU看，指令串行执行；

从PU看，数据并行执行

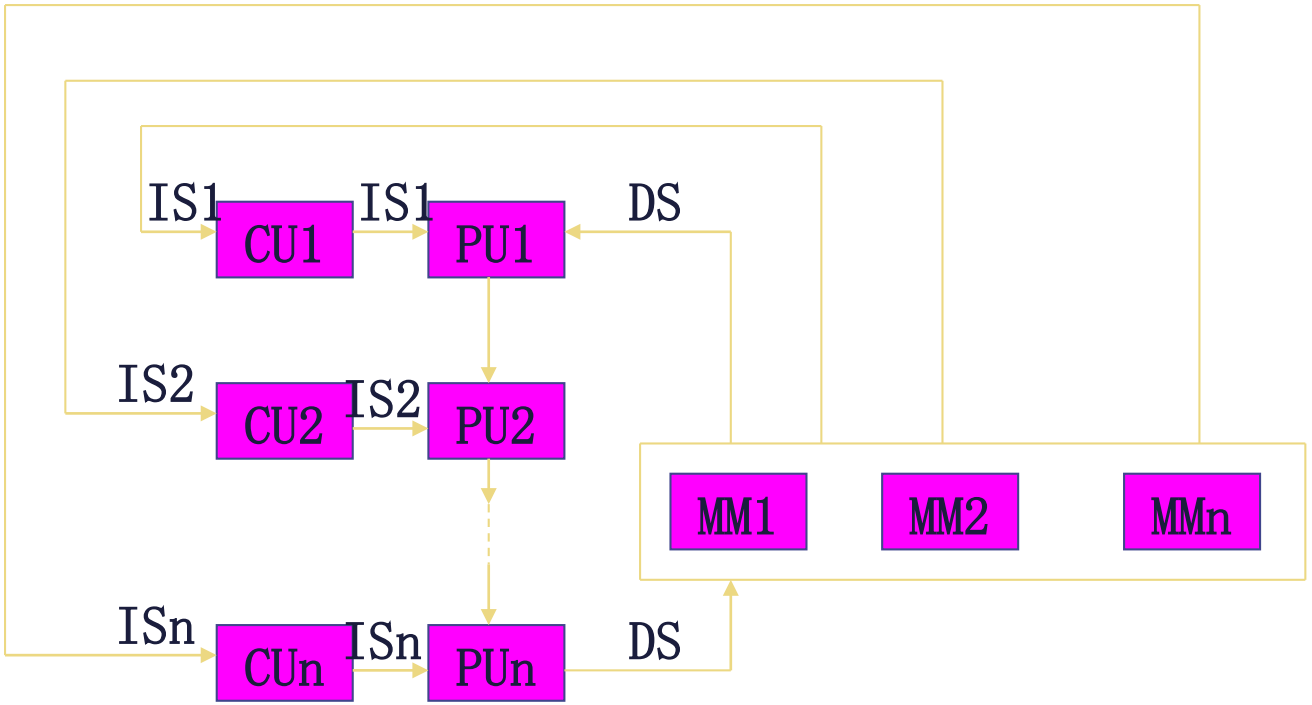
多指令流多数据流MIMS(Multiple Instruction Multiple Datastream)

MIMD：多处理机系统



多指令流单数据流MISD(Multiple Instruction Single Datastream)：

几条指令对同一个数据进行不同处理，实际上不存在



- 引言
- 计算机体系结构的概念/分类
- 定量分析技术基础
- 计算机体系结构的发展

1.2.1 性能设计和评测的基本原则

三条基本原则和方法：

1. 大概率事件优先的原则

对于大概率事件（最常见的事件），赋予它优先的处理权和资源使用权，以获得全局的最优结果。

2. Amdahl定律

加快某部件执行速度所获得的系统性能加速比，受限于该部件在系统中所占的重要性。

3. 程序局部性原理

程序在执行时所访问地址的分布不是随机的，而是相对地簇聚；这种簇聚包括指令和数据两部分。

- 程序的时间局部性：程序即将用到的信息很可能就是目前正在使用的信息。
- 程序的空间局部性：程序即将用到的信息很可能与目前正在使用的信息在空间上相邻或者临近。

（一）Amdahl定律：

系统中某一部件由于采用某种更快的执行方式后整个系统性能的提高与这种执行方式的使用频率或占总执行时间的比例有关。

(1) 加速比

$$\text{系统加速比} = \frac{\text{系统性能}_{\text{改进后}}}{\text{系统性能}_{\text{改进前}}} = \frac{\text{总执行时间}_{\text{改进前}}}{\text{总执行时间}_{\text{改进后}}}$$

系统加速比依赖于两个因素：

- 可改进比例：可改进部分在原系统计算时间中所占的比例，它总是小于等于 1 的。

例如：一个需运行60秒的程序中有20秒的运算可以加速，该比例为20/60。

- **部件加速比**：可改进部分改进以后的性能提高，一般情况下它是大于 1 的 。

例如：系统改进后执行程序，其中可改进部分花费2秒，而改进前该部分需5秒，则性能提高为5/2。

$$\text{总执行时间}_{\text{改进后}} = \text{不可改进部分的执行时间} + \text{可改进部分改进后的执行时间}$$

$$\text{总执行时间}_{\text{改进后}} = (1-\text{可改进比例}) \times \text{总执行时间}_{\text{改进前}} + \frac{\text{可改进比例} \times \text{总执行时间}_{\text{改进前}}}{\text{部件加速比}}$$

$$= \left[(1-\text{可改进比例}) + \frac{\text{可改进比例}}{\text{部件加速比}} \right] \times \text{总执行时间}_{\text{改进前}}$$

系统加速比为改进前与改进后总执行时间之比：

$$\text{系统加速比} = \frac{\text{总执行时间}_{\text{改进前}}}{\text{总执行时间}_{\text{改进后}}}$$

$$= \frac{1}{(1 - \text{可改进比例}) + \dfrac{\text{可改进比例}}{\text{部件加速比}}}$$

$$Fe = \frac{\text{可改进部分占用的时间}}{\text{改进前整个任务的执行时间}}$$ ，它总小于1。

$$Se = \frac{\text{改进前改进部分的执行时间}}{\text{改进后改进部分的执行时间}}$$ ，它总大于1。

改进后整个任务的执行时间为：

$$T_n = T_o \cdot (1 - Fe + \frac{Fe}{Se})$$

其中To为改进前的整个任务的执行时间。

改进后整个系统的加速比为：

$$S_n = \frac{To}{T_n} = \frac{1}{(1-Fe) + \frac{Fe}{Se}}$$

（1-Fe）表示不可改进部分。

当 $Se \to \infty$时，则$S_n = \frac{1}{1-Fe}$

可改进极限受Fe的约束。

Amdahl应用举例

1)设一个程序由A、B两部分组成，其中A为可并行化部分，可由多个CPU并行执行，A：B = 1：3，当将该程序运行于具有4 CPU的计算机上时，该程序的性能提高多少？

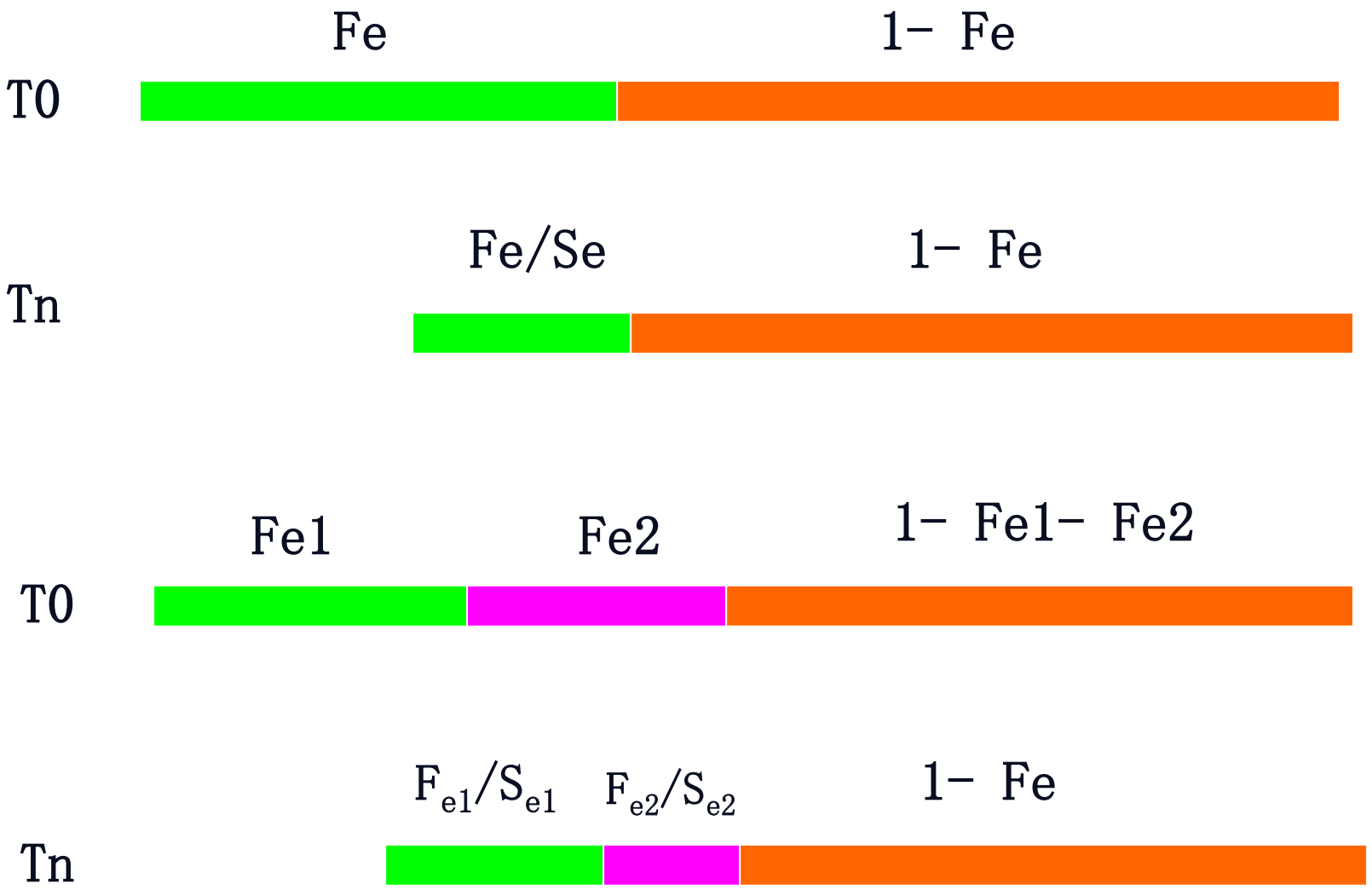
- Fe = ¼
- Se = 4

代入加速比公式：

$$S_n = \frac{T_o}{T_n} = \frac{1}{(1-F_e)+\frac{F_e}{S_e}}$$

$$= \frac{1}{(1-1/4)+\frac{1/4}{4}} = 16/13 = 1.23$$

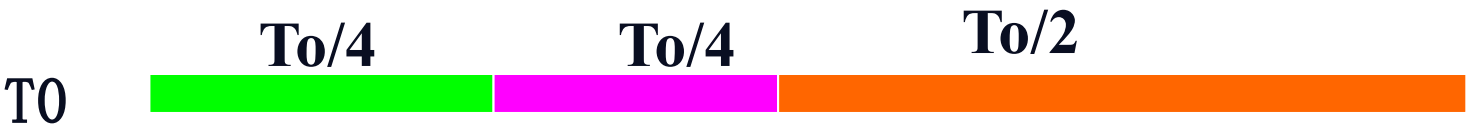
Amdahl的直观理解



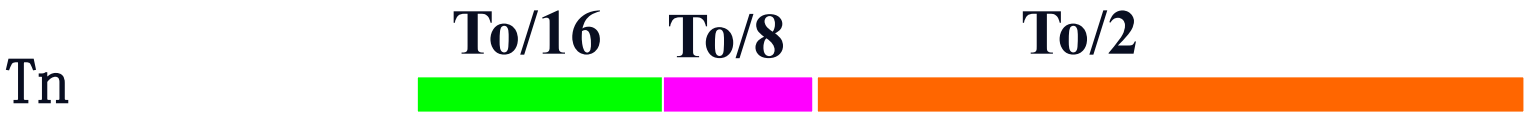
3)设一个程序由A、B、C三部分组成，其中A为可并行化部分，B为I/O部分，C为其它部分，且A：B：C = 1：1：2，当将该程序运行于具有4 CPU的计算机，且更换一个速度为原来速度2倍的磁盘时，该程序的性能提高多少？

•方法1 ：利用Amdahl的直观理解求解

设原来程序总执行时间为To，则A、B、C的时间分别为To/4、To/4、To/2。



更换环境后上述各部分时间比例如下图所示：



则加速比为：　　S = To / (To/16 + To/8 + To/2) = 16/11

•方法2：直接利用Amdahl定理求解

先假设不更换磁盘，则Fe1 = ¼ ，则

S1= 1/ (1- ¼ + 1/4/4) = 16/13

在上述条件下假设更换磁盘，此时原来A、B、C所占用比例为：

A’：B’：C’ = 1：4：8 Fe2 = 4/13

再次使用Amdahl定理，得S2=1/(1- 4/13 + 4/13/2) = 26/22

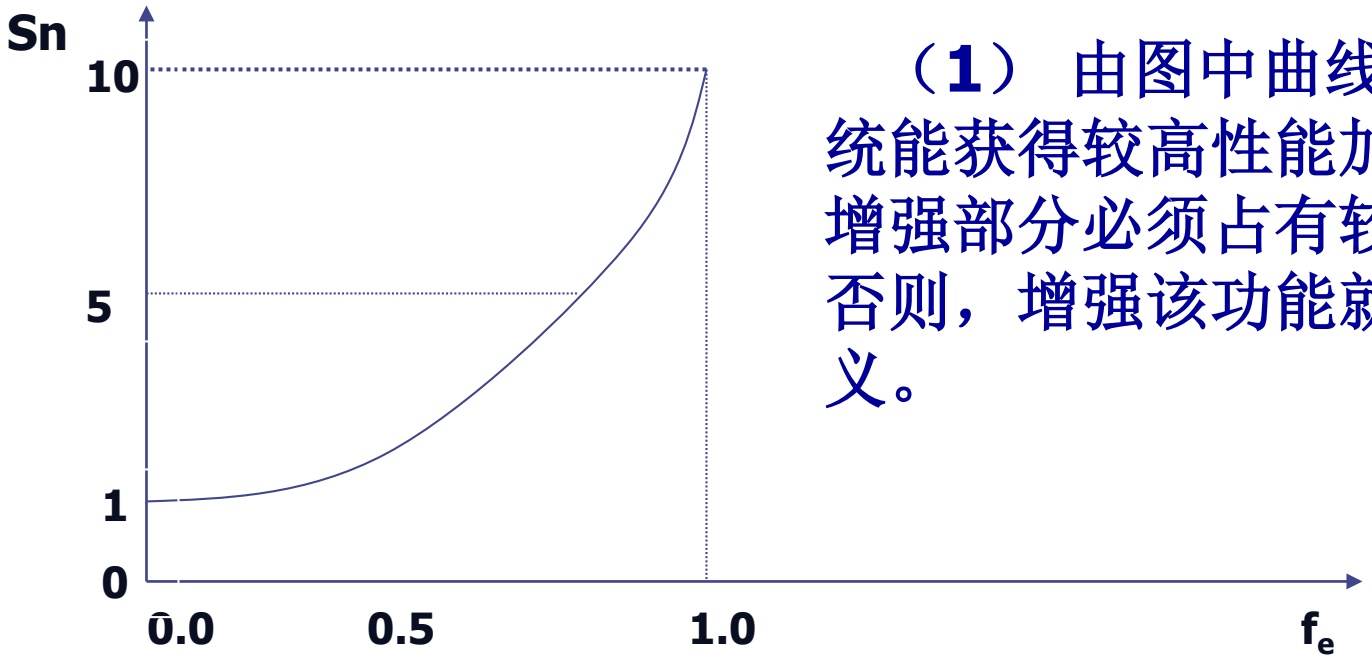
则总的加速比为 S = S1 * S2 = 16/13 * 26 /22 = 16/11

加速比Sn与可增强性能部分f_e的关系

例 若考虑将系统中某一功能的处理速度加快10倍，但该功能的处理使用时间仅为整个系统运行时间的40%，则采用此增强功能方法后，能使整个系统的性能提高多少？

已知：Fe=0.4，Se=10，则可得： Sn =1.56。

它说明局部（40%）大幅度改进（10倍）对全局作用要小得多（1.56倍）。



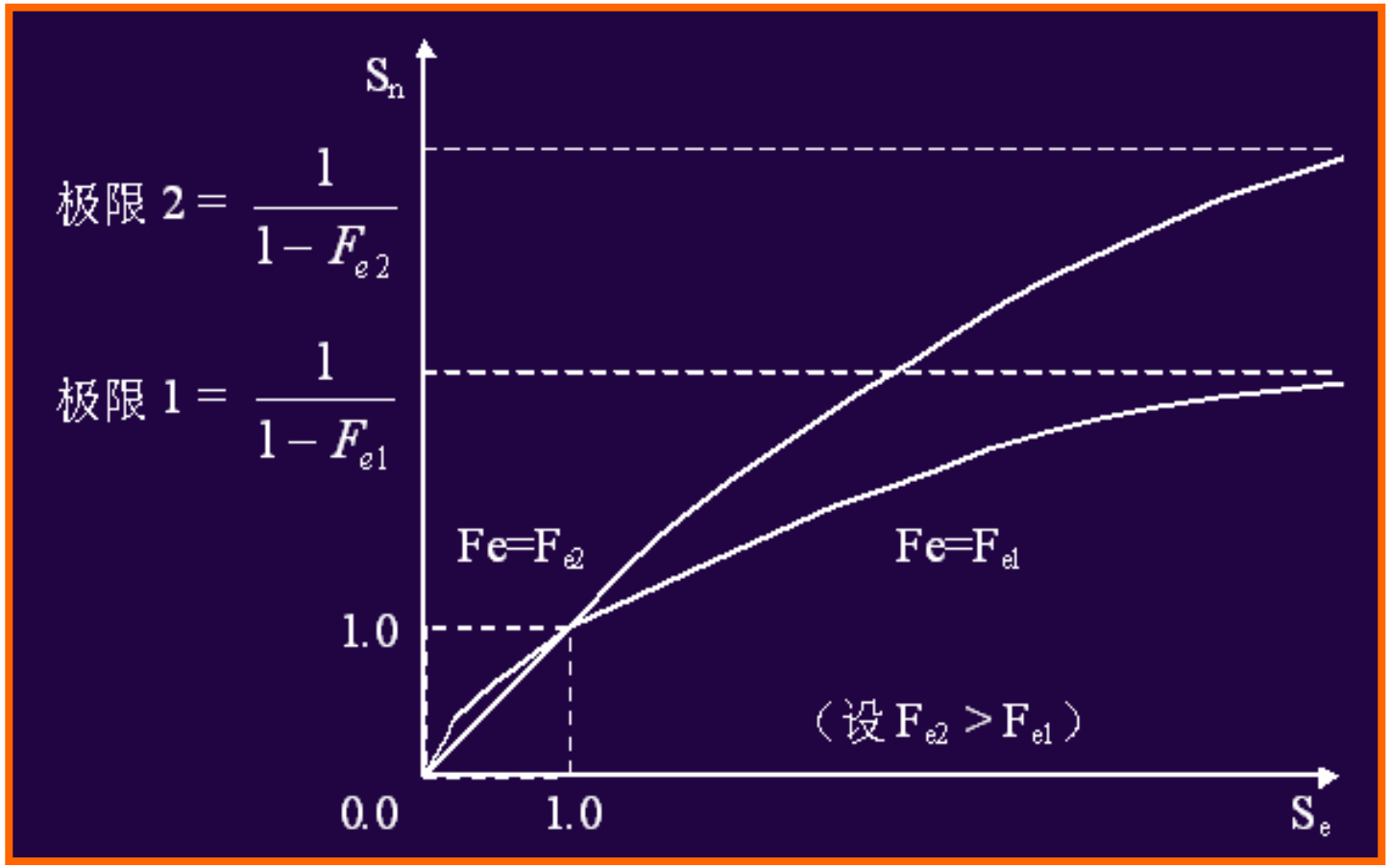
（1） 由图中曲线可知，为使系统能获得较高性能加速比，则可增强部分必须占有较大的比例；否则，增强该功能就没有多大意义。

(2) Amdahl定律：性能递减规则

如果仅仅对计算机中的一部分做性能改进，则改进越多，系统获得的效果越小。

推论：如果只针对整个任务的一部分进行优化，那么所获得的加速比不大于1/(1-fe)。

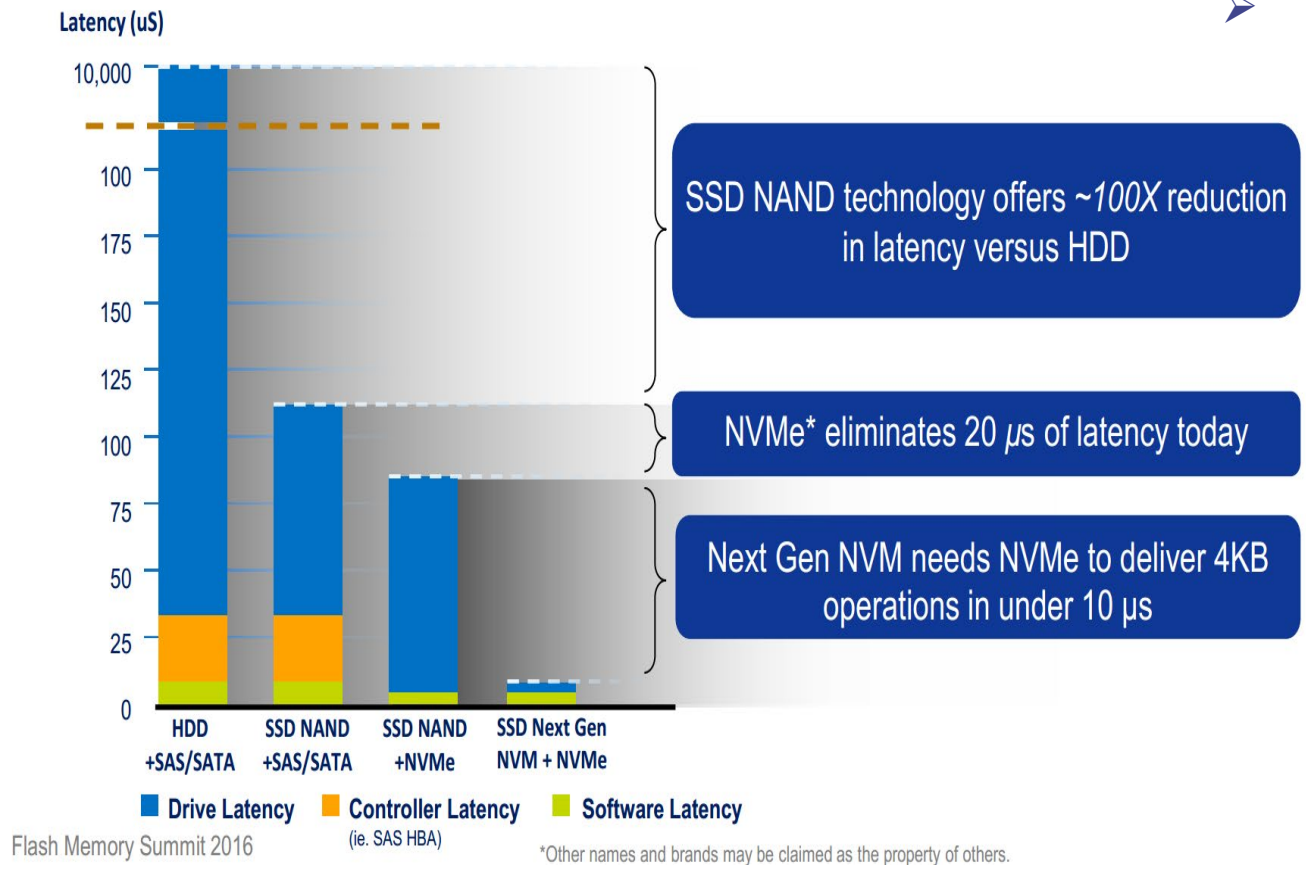
增大Se和Fe对Sn都有提升作用；但当Fe固定时，一味增大Se对Sn的作用会越来越不显著。



(3) 一个“好”的计算机系统：具有高性价比的计算机系统是一个带宽平衡的系统，而不是看它使用的某些部件的性能 。

实例应用

NVMe性能优势（总）

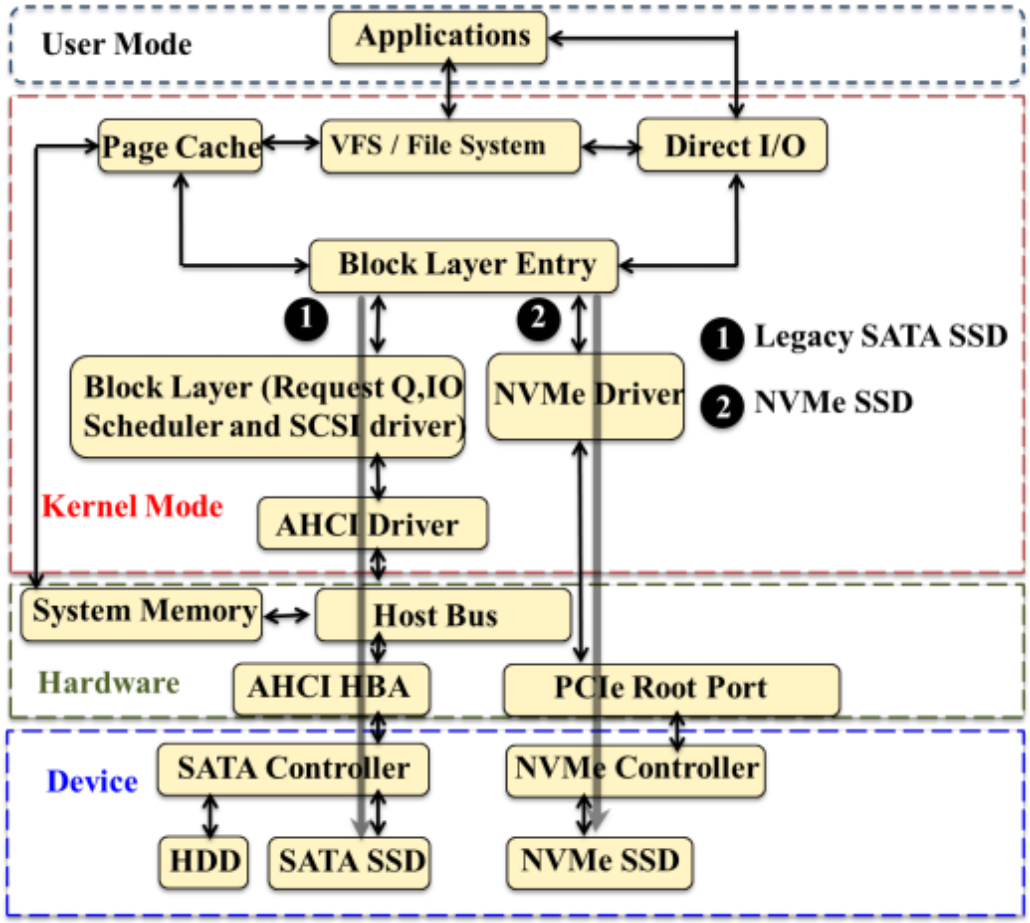


➢ **NVMe SSD的性能提高来自**

- 更短的IO软件路径
- 更多的IO队列
- MSI-X中断
- 很好的扩展性
- 更高的带宽
- 更少的通信开销
- 丰富的命令集

实例应用

NVMe SSD性能优势（一）---更短IO软件路径



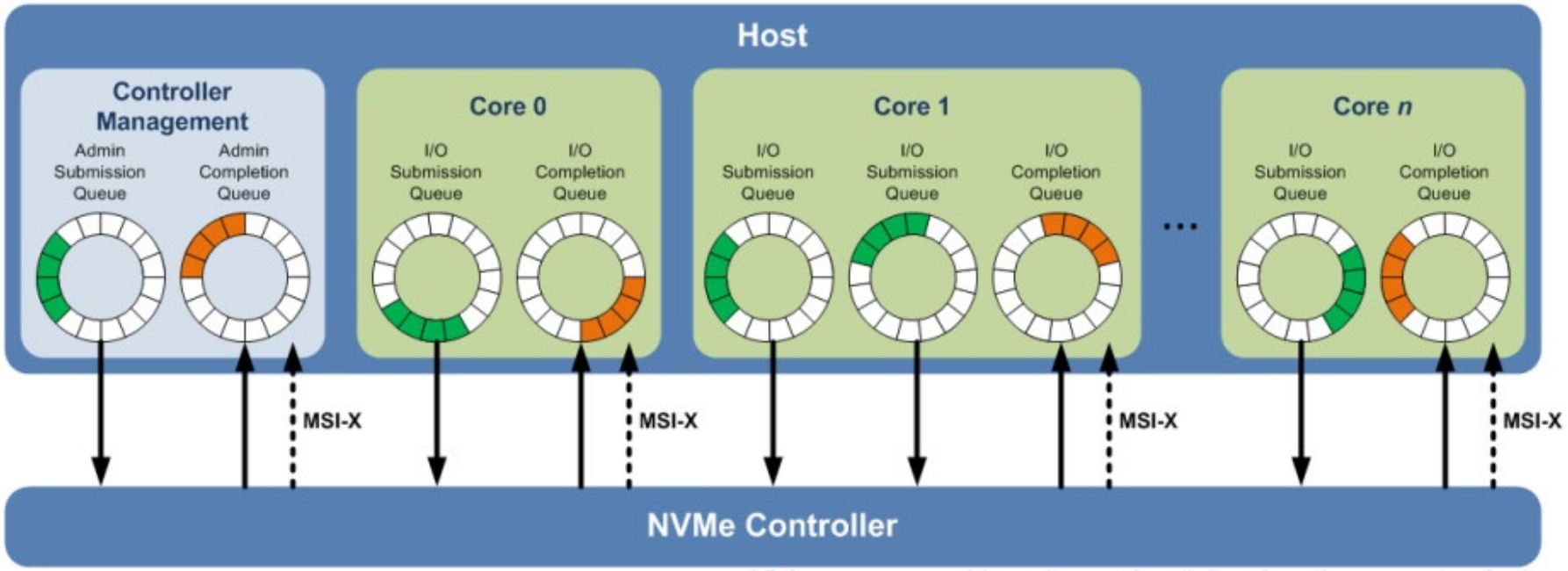
NVMe/PCIe SSD VS AHCI/SATA SSD软件路径的优势

- **NVMe软件路径相比SATA更短，减少了IO Scheduler(Linux 自带的IO调度主要是针对HDD特性设置的，并不适用于这种随机性能基本等于顺序读写性能的SSD)以及3层SCSI层（图中所示SCSI driver,减少了各层之间命令的转换时间以及管理时间)**
- **SATA接口与存储设备进行通信需要首先要经过Host Bus然后经过AHCI HBA，而PCIe接口与存储设备通信只需要通过PCIe Root Port即可。**

实例应用

NVMe 的优势（二）---多队列

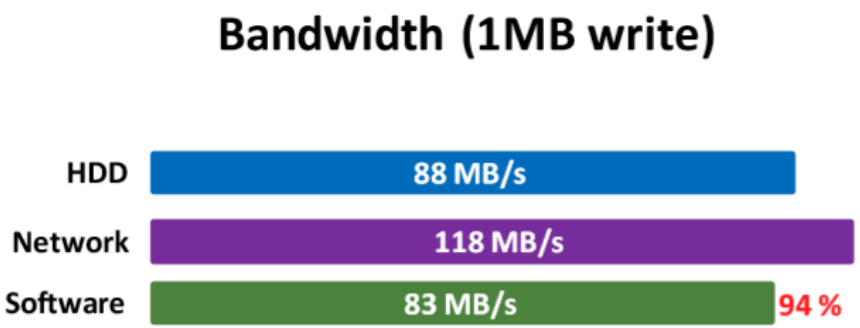
- NVMe驱动中包括一对Admin队列和多对IO队列。对于IO队列，SQ与CQ的关系可以是1：1或者多对1
  - 在多核的主机系统中，每个核都可以有自己的SQ和CQ,移除处理器之间的队列锁（提高带宽），此外还能维持每个请求的上下文，减少不必要上下文切换（减少延迟）。



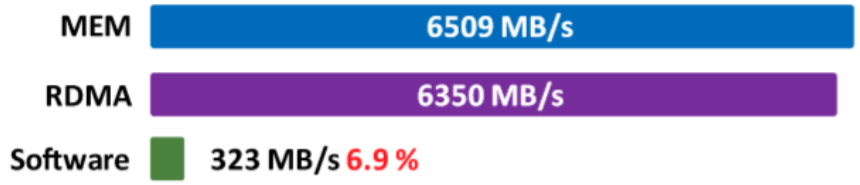
实例应用

Modular-Designed Distributed File System

- DiskGluster
  - Disk for data storage
  - GigE for communication



- MemGluster
  - Memory for data storage
  - RDMA for communication

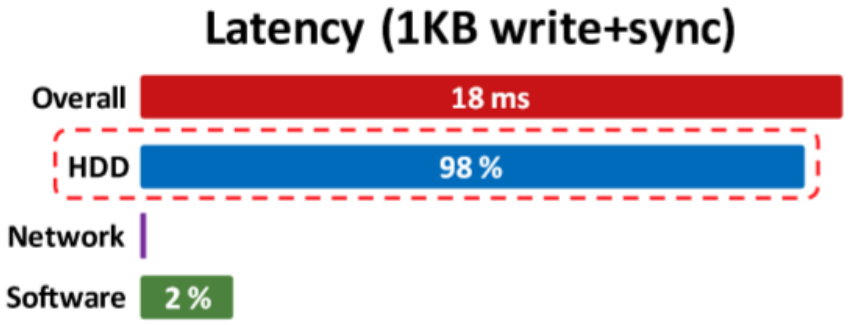


12

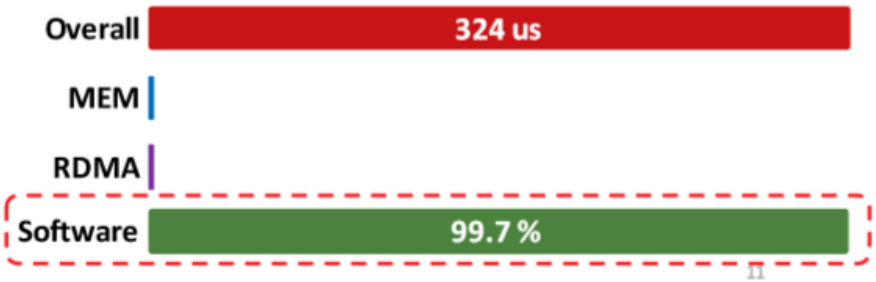
实例应用

Modular-Designed Distributed File System

- DiskGluster
  - Disk for data storage
  - GigE for communication



- MemGluster
  - Memory for data storage
  - RDMA for communication



（二） CPU的性能

1. 将程序执行的时间进行分解

(1) 计算机工作的时钟频率
计算机系统中与实现技术和工艺有关的因素。
单位是MHz(f)。

(2) 总时钟周期数

程序执行的cpu时间 ：

CPU时间 = 总时钟周期数 / 时钟频率

2. “指令时钟数”CPI(Cycles Per Instruction)

一个与计算机体系结构有关的参数。

CPI = 总时钟周期数 / IC

IC：程序执行过程中所处理的指令数。

3. 程序执行的CPU时间可以写成

总CPU时间 = CPI × IC / 时钟频率

总CPU时间 = CPI × IC × CT

◆ CPI：反映了计算机实现技术、计算机指令集结构和计算机组织。

◆ IC：反映了计算机指令集的结构和编译技术。

◆ 时钟频率/时钟周期时间 ：反映了计算机实现技术、生产工艺和计算机组织。

Processor Performance Optimizations

- Independent performance optimization technologies

  ---compiler

  ---faster circuit technology or an advanced fabrication process

Such technologies are desirable and should be employed if the cost is not prohibitive

Processor Performance Optimizations

- Dependent performance optimization technologies

---reduce the instruction count

first, the instruction set can include more complex instructions that perform more work per instruction.

Second, certain compiler optimizations can result in fewer instructions being executed. eg, unrolling loops,in-lining function call

Third, computation reuse techniques.

---reduce CPI

RISC; instruction pipeline; cache memory; supersalar

---reduce cycle time

Pipelining;

Processor Performance Optimizations

Conclusion:

Achieving a performance improvement is not a straightforward task. It requires interesting tradeoffs involving Many issues.

总CPU时间 = CPI × IC / 时钟频率

总CPU时间 = CPI × IC × CT

4. 对CPU性能公式进行进一步细化

假设：计算机系统有n种指令；

CPI_i ：第i种指令的处理时间；

IC_i ：在程序中第i种指令出现的次数；

则程序执行时间为

CPU时间 $= \sum_{i=1}^{n} (CPI_i \times IC_i) / $ 时钟频率

$CPI = \sum_{i=1}^{n} (CPI_i \times IC_i) / IC = \sum_{i=1}^{n} (CPI_i \times IC_i / IC)$

其中：(IC_i / IC)反映了第i种指令在程序中所占的比例。

CPI应用举例

根据CPU时间 = (IC×CPI)/时钟频率= (IC×CPI) × CT 和Amdahl定理，可以重新计算机系统的加速比：

$$S = \text{CPU时间o} / \text{CPU时间n} = \frac{\frac{IC_1 \times CPI_1}{f_1}}{\frac{IC_2 \times CPI_2}{f_2}}$$

$$= \frac{IC_1 \times CPI_1 \times f_2}{IC_2 \times CPI_2 \times f_1}$$

相同程序,IC不变

$$= \frac{CPI_1 \times f_2}{CPI_2 \times f_1}$$

例

如果FP操作的比例是25%，FP操作平均CPI=4.0,其它指令平均CPI为1.33，FPSQR操作比例为2%，FRSQR平均CPI=20。假设有两种设计方案，分别把FPSQR操作的CPI和所有FP操作的CPI减为2，用CPU性能公式比较这两种设计方案。

解：$CPI_o = 4\times25\% + 1.33 \times75\% = 2$

方案1：仅把FPSQR的CPI减为2，则此时的CPI为：

$$CPI1 = 2 - 2\% \times (20 - 2) = 1.64$$

方案2：把所有的FP的CPI都减为2，则

$$CPI2 = 2 - 25\% \times (4 - 2) = 1.5$$

综上所述：方案2更好。

例　假设我们考虑条件分支指令的两种不同设计方法如下：

（1）CPU_A：通过比较指令设置条件码，然后测试条件码进行分支。

（2）CPU_B：在分支指令中包括比较过程

两种CPU中，条件分支指令都占用2个时钟周期而所有其它指令占用1个时钟周期.对于CPU_A，执行指令中分支指令占20%；由于每个分支指令之前都需有比较指令，因此比较指令也占20%。由于CPU_A在分支时不需要比较，因此假设它的时钟周期时间比CPUB快1.25倍。哪一个CPU更快？如果CPUA的时钟周期时间仅仅比CPUB快1.1倍，哪一个CPU更快呢？

解：我们不考虑所有系统问题，所以可用CPU性能公式。占用2个时钟周期的分支指令占总指令的20%，剩下的指令占用1个时钟周期。所以

$$CPI_A = 0.2 \times 2 + 0.80 \times 1 = 1.2$$

则CPU性能为：

$$\text{总CPU时间}_A = IC_A \times 1.2 \times \text{时钟周期}_A$$

根据假设，有：

$$\text{时钟周期}_B = 1.25 \times \text{时钟周期}_A$$

在CPU_B中没有独立的比较指令，所以CPU_B的程序量为CPU_A的80%，分支指令的比例为：

$$20\%/80\% = 25\%$$

这些分支指令占用2个时钟周期，而剩下75%的指令占用1个时钟周期，因此：

$$CPI_B = 0.25 \times 2 + 0.75 \times 1 = 1.25$$

因为CPU_B不执行比较，故：

$$IC_B = 0.8 \times IC_A$$

因此CPU_B性能为：

$$\text{总CPU时间}_B = IC_B \times CPI_B \times \text{时钟周期}_B$$

$$= 0.8 \times IC_A \times 1.25 \times (1.25 \times \text{时钟周期}_A)$$

$$= 1.25 \times IC_A \times \text{时钟周期}_A$$

$$\text{总CPU时间}_A = IC_A \times 1.2 \times \text{时钟周期}_A$$

在这些假设之下，尽管CPU_B执行指令条数较少，CPU_A因为有着更短的时钟周期，所以比CPU_B快。

如果CPU_A的时钟周期时间仅仅比CPU_B快1.1倍，则

$$时钟周期_B = 1.10 \times 时钟周期_A$$

CPU_B的性能为：

$$总CPU时间_B = IC_B \times CPI_B \times 时钟周期_B$$

$$= 0.8 \times IC_A \times 1.25 \times (1.10 \times 时钟周期_A)$$

$$= 1.10 \times IC_A \times 时钟周期_A$$

因此CPU_B由于执行更少指令条数，比CPU_A运行更快。

（三） 计算机性能的评测

衡量计算机的最主要指标是性能、成本

性能指标：

响应时间（response time）—— 完成一个任务的全部时间，包括磁盘访问时间、存储器访问时间、I/O访问时间等。（从用户的角度看问题）

吞吐率（throughout） —— 单位时间内完成的任务数，Bytes/S从系统管理员的角度看问题）

相同点：都认为能够以最短时间完成指定任务的计算机就是最快的。

不同点：响应时间针对单任务，而流量针对多任务。

测试程序

1. 目前常用的测试程序可以分为五类：

（按测试可靠性由高至低的顺序列出）

(1) 实际应用程序

(2) 修正的（或者脚本化）应用程序

(3) 核心测试程序

(3) 小测试程序

(4) 合成测试程序

测试程序

2. 测试程序组件

选择一组各个方面有代表性的测试程序，组成通用测试程序集合。

最大优点：避免独立测试程序存在的片面性，尽可能全面地测试计算机系统的性能。

- 最常见的测试程序组件是基于UNIX的SPEC

主要版本包括SPEC89、SPEC92、SPEC95和SPEC2000等。

其他性能评价指标

1、 性能评测的常用方法以运算速度为主

2、 时钟频率（主频）：用于同类处理机之间

如：PentiumⅡ/450比PentiumⅡ/300快50%，...

3、指令执行速度

MIPS（Million Instructions Per Second）,KIPS,GIPS,TIPS

$$MIPS = \frac{\text{指令条数}}{\text{执行时间} \times 10^6} = \frac{\text{指令条数}}{\text{CPU时钟周期数} \times 10^6 / f} = \frac{f}{CPI \times 10^6}$$

$$\text{程序的执行时间}Te = \frac{\text{指令条数}}{MIPS \times 10^6}$$

其他性能评价指标

衡量机器性能的唯一可靠的标准就是真正的执行程序的时间

$$MIPS = \frac{IC}{T_e} \times 10^{-6} = \frac{IC}{IC \times CPI \times CYCLE} \times 10^{-6} = \frac{f}{CPI} \times 10^{-6}$$ ，主要用于标量计算机；

例子：PentiumII 450的CPI＝0.5，计算该处理机的运算速度

解： 已知 CPI ＝0.5　　f = 450MHz

MIPS = f×10^6/CPI = 450×10^6×10^{-6}/0.5 = 900MIPS

其他性能评价指标

MIPS的三个缺陷：

a）MIPS依赖于指令集，所以用它来比较指令集不同的机器的性能好坏是不准确的。

b）在同一台机器上，它因程序不同而变化。

c）它可能与性能相反。如具有可选硬件浮点运算部件的机器。

其他性能评价指标

MFLOPS：Million Floating Operation Per Second

每秒百万次浮点操作次数

$$MFLOPS = \frac{\text{程序中的浮点操作次数}}{\text{执行时间}\times 10^6}$$ ，主要用于向量计算机。

分析： MFLOPS只能用来衡量机器浮点操作的性能，而不能体现机器的总体性能。

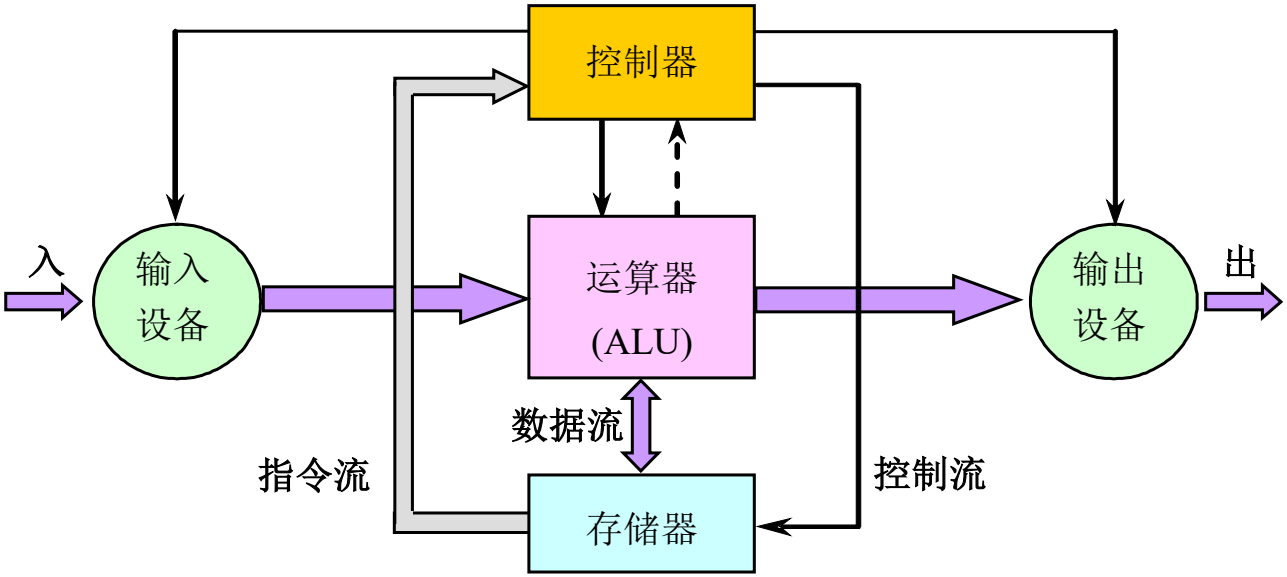
优点：基于操作而非指令，可以用来比较两种不同的机器。

缺点：它并不可靠，因为不同的机器上浮点运算集不同；依赖于操作类型；单个程序的值并不能反映机器的性能。

- 引言
- 计算机体系结构的概念/分类
- 定量分析技术基础
- 计算机体系结构的发展

1.4 计算机系统结构的发展

1.4.1 冯·诺依曼结构及其改进



存储程序计算机的结构

1. 存储程序原理的基本点：指令驱动

程序预先存放在计算机存储器中，机器一旦启动，就能按照程序指定的逻辑顺序执行这些程序，自动完成由程序所描述的处理工作。

2. 冯·诺依曼结构的主要特点

- 计算机以运算器为中心。
- 在存储器中，指令和数据同等对待。

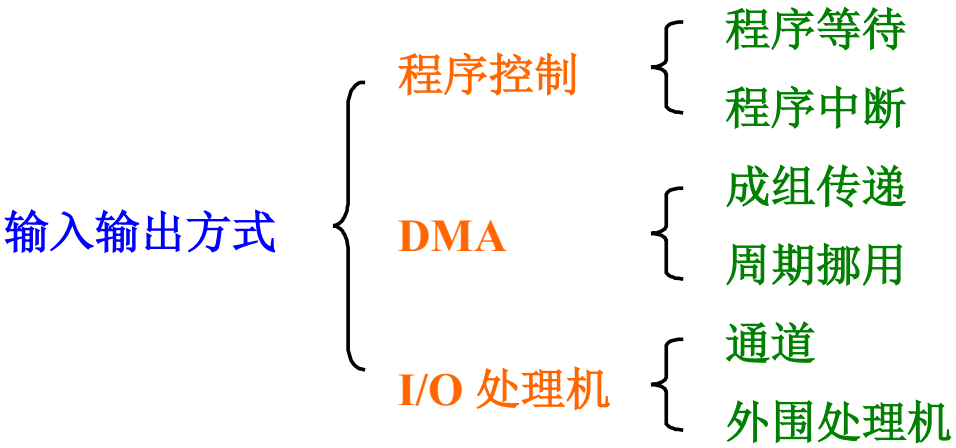
指令和数据一样可以进行运算，即由指令组成的程序是可以修改的。

- 存储器是按地址访问、按顺序线性编址的一维结构，每个单元的位数是固定的。

- 指令的执行是顺序的。
  - 一般是按照指令在存储器中存放的顺序执行。
  - 程序的分支由转移指令实现。
  - 由指令计数器PC指明当前正在执行的指令在存储器中的地址。
- 指令由操作码和地址码组成。
- 指令和数据均以二进制编码表示，采用二进制运算。

3. 对系统结构进行的改进

➢ 输入/输出方式的改进



➢ 采用并行处理技术

- 如何挖掘传统机器中的并行性？
- 在不同的级别采用并行技术。

  例如：微操作级、指令级、线程级、进程级、任务级等。

- 存储器组织结构的发展
  - 相联存储器与相联处理机
  - 通用寄存器组
  - 高速缓冲存储器Cache
- 指令系统的发展

  两个发展方向：
  - 复杂指令集计算机CISC
  - 精减指令集计算机RISC

1.5 计算机系统结构中并行性的发展

1.5.1 并行性的概念

1. 并行性：计算机系统在同一时刻或者同一时间间隔内进行多种运算或操作。

只要在时间上相互重叠，就存在并行性。

- 同时性：两个或两个以上的事件在同一时刻发生。
- 并发性：两个或两个以上的事件在同一时间间隔内发生。

2. 从处理数据的角度来看，并行性等级从低到高可分为：

- 字串位串：每次只对一个字的一位进行处理。
  最基本的串行处理方式，不存在并行性。
- 字串位并：同时对一个字的全部位进行处理，不同字之间是串行的。
  开始出现并行性。
- 字并位串：同时对许多字的同一位（称为位片）进行处理。
  具有较高的并行性。
- 全并行：同时对许多字的全部位或部分位进行处理。
  最高一级的并行。

3. 从执行程序的角度来看，并行性等级从低到高可分为：

- 指令内部并行：单条指令中各微操作之间的并行。
- 指令级并行：并行执行两条或两条以上的指令。
- 线程级并行：并行执行两个或两个以上的线程。

  通常是以一个进程内派生的多个线程为调度单位。
- 任务级或过程级并行：并行执行两个或两个以上的过程或任务（程序段）

  以子程序或进程为调度单元。
- 作业或程序级并行：并行执行两个或两个以上的作业或程序。

1.5.2 提高并行性的技术途径

三种途径：

时间重叠

引入时间因素，让多个处理过程在时间上相互错开，轮流重叠地使用同一套硬件设备的各个部分，以加快硬件周转而赢得速度。

资源重复

引入空间因素，以数量取胜。通过重复设置硬件资源，大幅度地提高计算机系统的性能。

资源共享

这是一种软件方法，它使多个任务按一定时间顺序轮流使用同一套硬件设备。

1.5.3 单机系统中并行性的发展

- 在发展高性能单处理机过程中，起主导作用的是时间重叠原理。
  - 实现时间重叠的基础：部件功能专用化

    把一件工作按功能分割为若干相互联系的部分；

    把每一部分指定给专门的部件完成；

    然后按时间重叠原理把各部分的执行过程在时间上重叠起来，使所有部件依次分工完成一组同样的工作。

- 在单处理机中，资源重复原理的运用也已经十分普遍。
  - 多体存储器
  - 多操作部件

    通用部件被分解成若干个专用部件，如加法部件、乘法部件、除法部件、逻辑运算部件等，而且同一种部件也可以重复设置多个。

    只要指令所需的操作部件空闲，就可以开始执行这条指令（如果操作数已准备好的话）。这实现了指令级并行。

➢ 阵列处理机（并行处理机）

更进一步，设置许多相同的处理单元，让它们在同一个控制器的指挥下，按照同一条指令的要求，对向量或数组的各元素同时进行同一操作，就形成了阵列处理机 。

1. 在单处理机中，资源共享的概念实质上是用单处理机模拟多处理机的功能，形成所谓虚拟机的概念。

- 分时系统

1.5.4 多机系统中并行性的发展

1. 多机系统遵循时间重叠、资源重复、资源共享原理，发展为3种不同的多处理机：

   同构型多处理机、异构型多处理机、分布式系统

2. 耦合度

   反映多机系统中各机器之间物理连接的紧密程度和交互作用能力的强弱。

   - 紧密耦合系统（直接耦合系统）：在这种系统中，计算机之间的物理连接的频带较高，一般是通过总线或高速开关互连，可以共享主存。

- 松散耦合系统（间接耦合系统）：一般是通过通道或通信线路实现计算机之间的互连，可以共享外存设备（磁盘、磁带等）。机器之间的相互作用是在文件或数据集一级上进行。

  表现为两种形式：

  - **多台计算机和共享的外存设备连接，不同机器之间实现功能上的分工（功能专用化），机器处理的结果以文件或数据集的形式送到共享外存设备，供其它机器继续处理。**
  - **计算机网，通过通信线路连接，实现更大范围的资源共享。**

3. 功能专用化（实现时间重叠 ）

- 专用外围处理机

  例如：输入/输出功能的分离

- 专用处理机

  如数组运算、高级语言翻译、数据库管理等，分离出来。

- 异构型多处理机系统

  由多个不同类型、至少担负不同功能的处理机组成，它们按照作业要求的顺序，利用时间重叠原理，依次对它们的多个任务进行加工，各自完成规定的功能动作。

4. 机间互连

- 容错系统
- 可重构系统

对计算机之间互连网络的性能提出了更高的要求。高带宽、低延迟、低开销的机间互连网络是高效实现程序或任务一级并行处理的前提条件。

- 同构型多处理机系统

由多个同类型或至少担负同等功能的处理机组成，它们同时处理同一作业中能并行执行的多个任务。

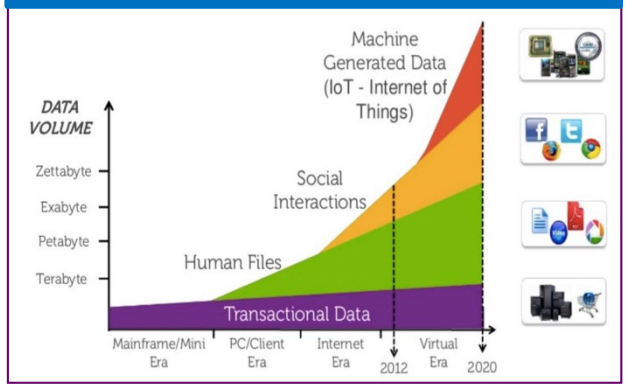
计算机系统结构的总趋势

按照数据的使用过程，计算机朝着3个方向发展。

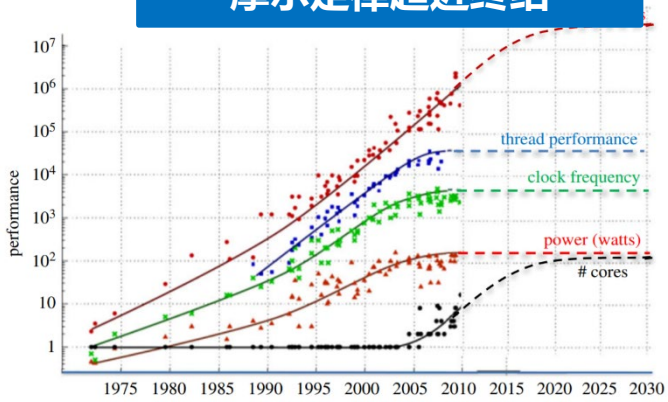
- 加快处理的速度
- 加快传输的速度
- 加快存储的速度和加大存储的容量
- 加快：1、器件快

  2、结构快（系统结构研究的课题）

计算数据量的爆炸性增长



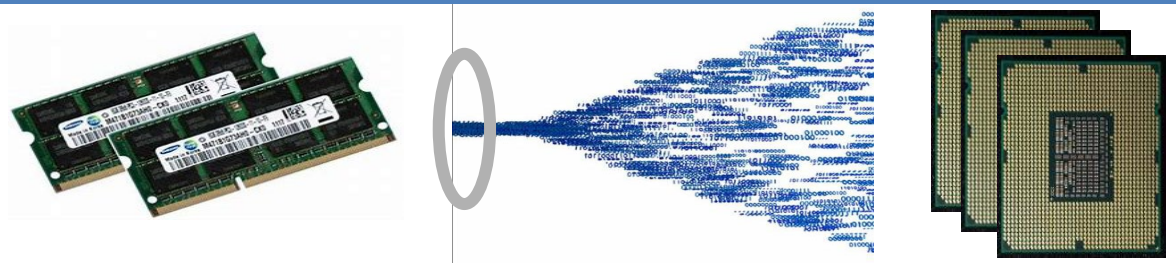
摩尔定律趋近终结



存储器与处理器间
大量的数据搬运

制程微缩带来的
能效红利结束

大量的数据移动——加剧的“冯诺依曼瓶颈”



大数据

物联网

人工智能

存算一体芯片：高性能、高能效、能应对大数据和IoT时代的数据洪流处理需求

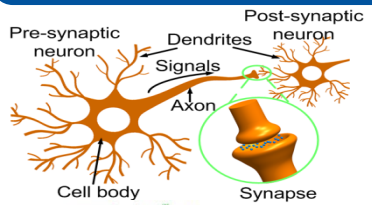
数据存储的挑战性问题需要从新存储器件与设备、新存储计算体系结构、新数据组织模式等多个层面协同解决

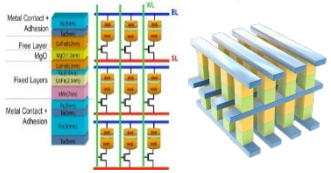
热数据

温数据

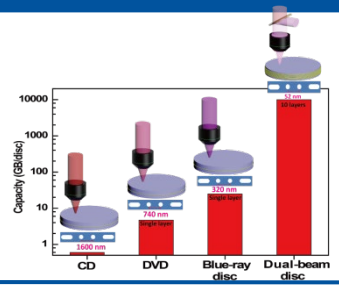
冷数据

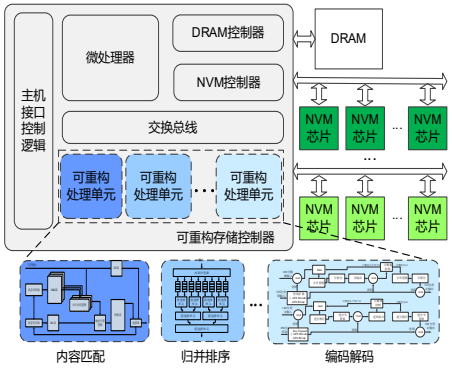
新型非易失存储新机理及器件



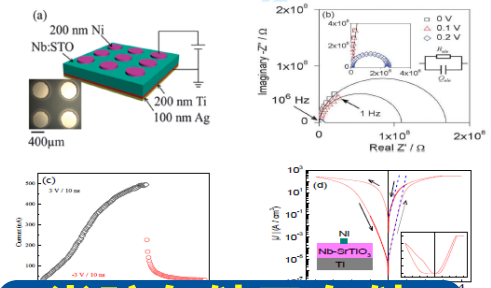


下一代超高密度光磁存储技术





存储计算融合新型存储体系结构



类脑存储及存储智能化

分布式数据组织及系统

器件与设备　　体系结构　　数据组织模式

本章小结

本章从定性知识和定量知识两个方面介绍计算机系统结构的基本概念。有关重点如下：

(1) 计算机系统结构的广义定义与狭义定义，与组织、实现的关系 ；
(2) 计算机系统的多级层次模型（6级），透明性概念；
(3) 冯.诺依曼型机器的特点，体系结构发展；
(4) 并行等级与方式

(5) Amdahl定律；
(6) ＣＰＵ时间；
(7) 平均周期数CPI
(8)每秒百万指令数MIPS

习题：P29，题1.7、1.9、1.10、1.11（分别用Amdahl公式和CPI公式求解）；
补充题：举例说明CPI、IC、CT相互影响的例子。