第1章 计算机系统结构的基础知识

张晨曦 刘依

www.GotoSchool.net

xzhang2000@sohu.com

1.1 计算机系统结构的基本概念

1. 第一台通用电子计算机诞生于1946年
2. 计算机技术的飞速发展受益于两个方面
   - 计算机制造技术的发展
   - 计算机系统结构的创新
3. 经历了四个发展过程

时 间	原 因	每年的性能增长
1946年起的25年	两种因素都起着主要的作用	25%
20世纪70年代末～80年代初	大规模集成电路和微处理器出现，以集成电路为代表的制造技术的发展	约35%
80年代中开始	RISC结构的出现，系统结构不断更新和变革，制造技术不断发展	50%以上 维持了约16年
2002年以来	3个（见下页）	约30%

- 大功耗问题；
- 可以进一步有效地开发的指令级并行性已经很少；
- 存储器访问速度的提高缓慢。

系统结构的重大转折：

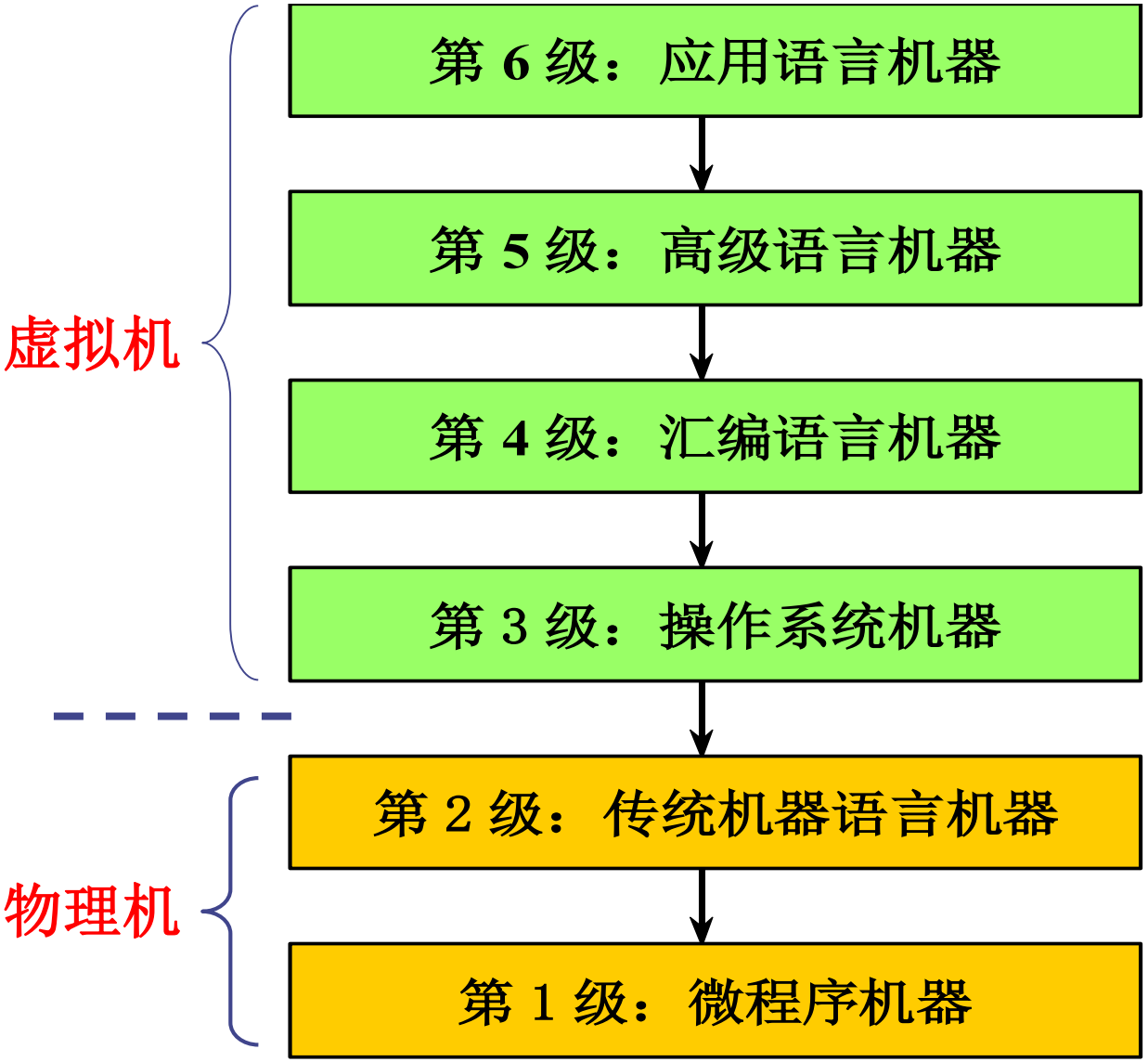
从单纯依靠指令级并行转向开发线程级并行和数据级并行。

计算机系统结构在计算机的发展中有着极其重要的作用。

1.1.1 计算机系统的层次结构

1. 计算机系统＝硬件/固件＋软件
2. 计算机语言从低级向高级发展

   高一级语言的语句相对于低一级语言来说功能更强，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。
   - 每一层以一种语言为特征


第 6 级：应用语言机器
第 5 级：高级语言机器
第 4 级：汇编语言机器
第 3 级：操作系统机器
虚拟机
第 2 级：传统机器语言机器
第 1 级：微程序机器
物理机

- 物理机：用硬件/固件实现的机器

  （最下面的两级机器）

- 虚拟机：由软件实现的机器
  - 虚拟机中有些操作可以由硬件或固件实现。
  - 固件：具有软件功能的硬件。
- 各机器级的实现主要靠翻译或解释，或两者的结合。
  - 翻译：先用转换程序把高一级机器上的程序转换为低一级机器上等效的程序，然后再在这低一级机器上运行，实现程序的功能。
  - 解释：对于高一级机器上的程序中的每一条语句或指令，都是转去执行低一级机器上的一段等效程序。执行

完后，再去高一级机器取下一条语句或指令，再进行解释执行，如此反复，直到解释执行完整个程序。

解释执行比编译后再执行所花的时间多，但占用的存储空间较少。

1.1.2 计算机系统结构的定义

1. 计算机系统结构的经典定义

传统机器程序员所看到的计算机属性，即概念性结构与功能特性。

（1964年 Amdahl在介绍IBM360系统时提出的）

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的属性。
3. 透明性

   在计算机技术中，把这种本来存在的事物或属性，但从某种角度看又好像不存在的概念称为透明性。

4. 广义的系统结构定义：指令系统结构、组成、硬件

   （计算机设计的3个方面）

计算机系统结构的实质：

确定计算机系统中软硬件的界面，界面之上是软件实现的功能，界面之下是硬件和固件实现的功能。

1.1.3 计算机组成和计算机实现

1. 计算机系统结构：计算机系统的软、硬件的界面

   即机器语言程序员所看到的传统机器级所具有的属性。

2. 计算机组成：计算机系统结构的逻辑实现

- 包含物理机器级中的数据流和控制流的组成以及逻辑设计等。
- 着眼于：物理机器级内各事件的排序方式与控制方式、各部件的功能以及各部件之间的联系。

3. 计算机实现：计算机组成的物理实现
   - 包括处理机、主存等部件的物理结构，器件的集成度和速度，模块、插件、底板的划分与连接，信号传输，电源、冷却及整机装配技术等。
   - 着眼于：器件技术（起主导作用）、微组装技术。

具有相同系统结构的计算机可以采用不同的计算机组成。

同一种计算机组成又可以采用多种不同的计算机实现。

举例：乘法指令 、主存容量与编址方式

4. 系列机

由同一厂家生产的具有相同系统结构、但具有不同组成和实现的一系列不同型号的机器。

例如 IBM公司的IBM370系列，Intel公司的x86系列等。

1.1.4 计算机系统结构的分类

常见的计算机系统结构分类法有3种：

Flynn分类法、冯氏分类法和Handler分类法

1. Flynn分类法

- 按照指令流和数据流的多倍性进行分类。
  - 指令流：计算机执行的指令序列
  - 数据流：由指令流调用的数据序列
  - 多倍性：在系统最受限的部件上，同时处于同一执行阶段的指令或数据的最大数目。

- 把计算机系统的结构分为4类
  - 单指令流单数据流SISD

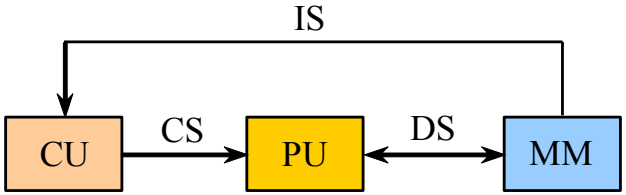
    （Single Instruction stream Single Data stream）
  - 单指令流多数据流SIMD

    （Single Instruction stream Multiple Data stream）
  - 多指令流单数据流MISD

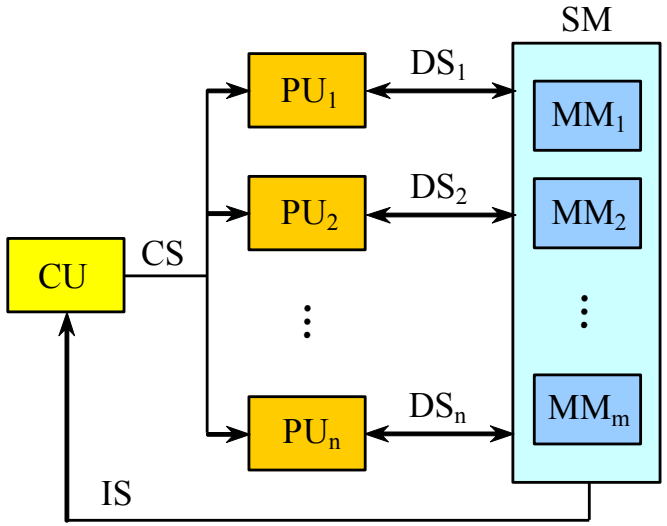
    （Multiple Instruction stream Single Data stream）
  - 多指令流多数据流MIMD

    （Multiple Instruction stream Multiple Data stream）

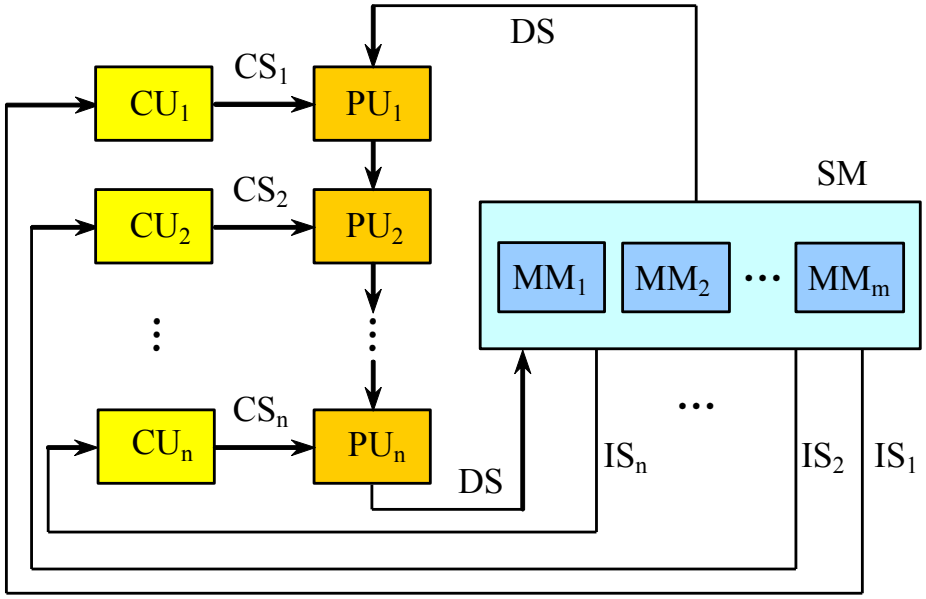
- 4类计算机的基本结构
  - IS：指令流
  - DS：数据流
  - CS：控制流
  - CU：控制部件
  - PU：处理部件
  - MM和SM：存储器



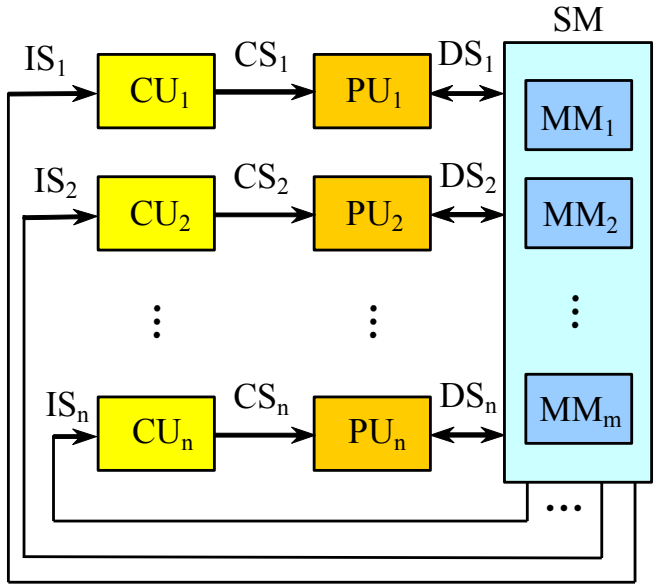
（a）SISD 计算机



（b）SIMD 计算机



（c）MISD 计算机

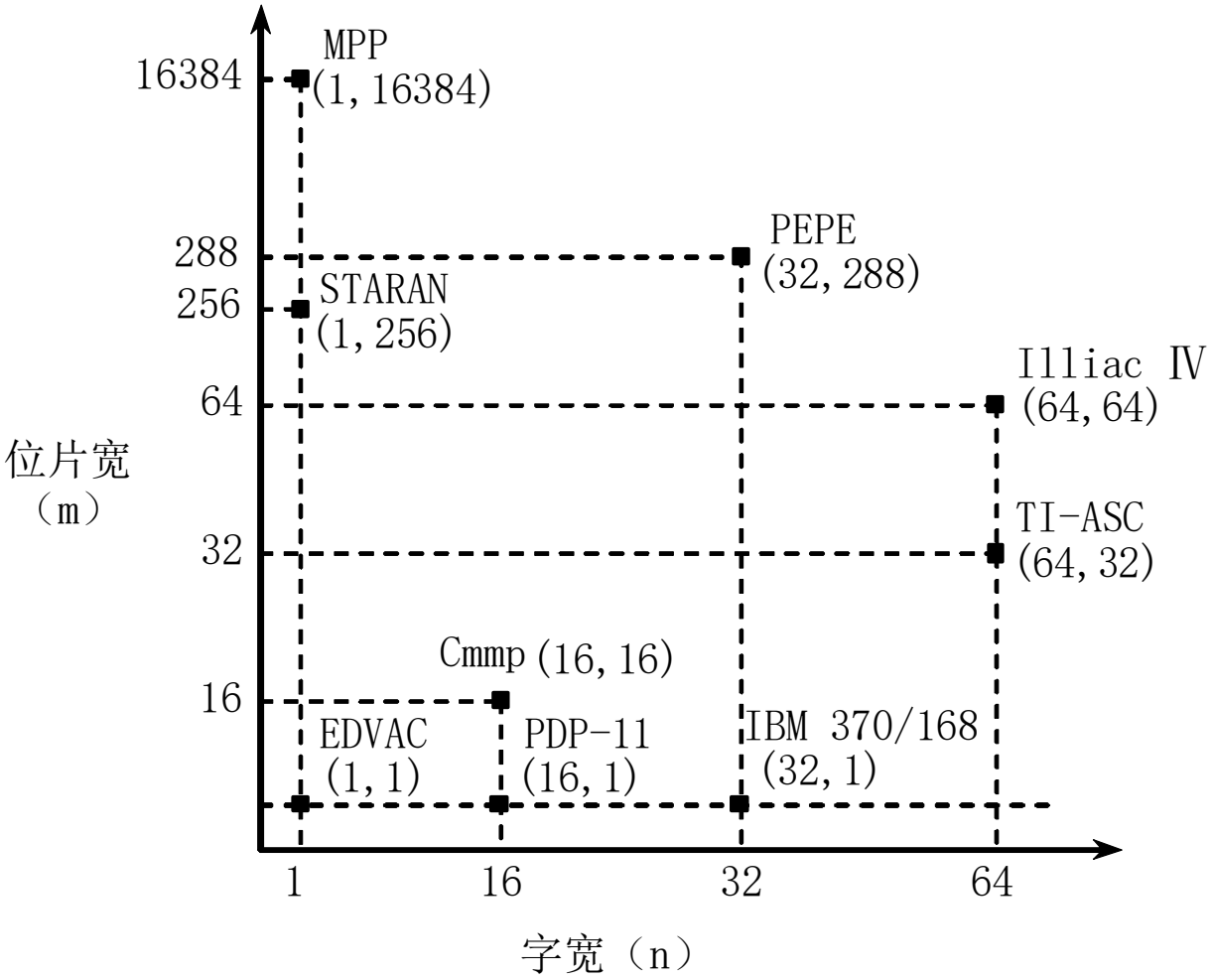


（d）MIMD 计算机

2. 冯氏分类法

- 用系统的最大并行度对计算机进行分类。
- 最大并行度：计算机系统在单位时间内能够处理的最大的二进制位数。

用平面直角坐标系中的一个点代表一个计算机系统，其横坐标表示字宽（n位），纵坐标表示一次能同时处理的字数（m字）。m×n就表示了其最大并行度。


16384
288
256
64
32
16
位片宽
（m）
MPP
(1, 16384)
PEPE
(32, 288)
STARAN
(1, 256)
Illiac Ⅳ
(64, 64)
TI-ASC
(64, 32)
Cmmp (16, 16)
EDVAC
(1, 1)
PDP-11
(16, 1)
IBM 370/168
(32, 1)
1
16
32
64
字宽（n）

- 4类不同最大并行度的计算机系统结构
  - 字串位串：n＝1，m＝1。
    (第一代计算机发展初期的纯串行计算机)
  - 字串位并： n＞1，m＝1。这是传统的单处理机，同时处理单个字的多个位，如16位、32位等。
  - 字并位串：n＝1，m＞1。同时处理多个字的同一位（位片）。
  - 字并位并：n＞1，m＞1。同时处理多个字的多个位。
- 平均并行度

  与最大并行度密切相关的一个指标。

  取决于系统的运用程度，与应用程序有关。

假设每个时钟周期内能同时处理的二进制位数为P_i，则T个时钟周期内的平均并行度为：

$$P_a = \frac{\sum_{i=1}^{T} P_i}{T}$$

系统在T个时钟周期内的平均利用率定义为：

$$\mu = \frac{P_a}{P_m} = \frac{\sum_{i=1}^{T} P_i}{TP_m}$$

3. Handler分类法

- 根据并行度和流水线对计算机进行分类。
- 把计算机的硬件结构分成3个层次
  - 程序控制部件（PCU）的个数k
  - 算术逻辑部件（ALU）或处理部件（PE）的个数d
  - 每个算术逻辑部件包含基本逻辑线路(ELC)的套数w
- 用公式表示

$$t（系统型号）=（k，d，w）$$

- 进一步改进

$$t（系统型号）=（k\times k'，d\times d'，w\times w'）$$

- k’：宏流水线中程序控制部件的个数
- d’：指令流水线中算术逻辑部件的个数
- w’：操作流水线中基本逻辑线路的套数

例如：Cray-1有1个CPU，12个相当于ALU或PE的处理部件，可以最多实现8级流水线。字长为64位，可以实现1～14位流水线处理。所以Cray-1系统结构可表示为：

t（Cray-1）＝(1，12×8，64×（1～14）)

几个例子：

t（PDP-11）＝(1，1，16)
t（Illiac Ⅳ）＝(1，64，64)
t（STARAN）＝(1，8192，1)
t（Cmmp）＝(16，1，16)
t（PEPE）＝(1×3，288，32)
t（TI-ASC）＝(1，4，64×8)

1.2 计算机系统的设计

1.2.1 计算机系统设计的定量原理

4个定量原理：

1. 以经常性事件为重点

- 对经常发生的情况采用优化方法的原则进行选择，以得到更多的总体上的改进。
- 优化是指分配更多的资源、达到更高的性能或者分配更多的电能等。

2. Amdahl定律

加快某部件执行速度所能获得的系统性能加速比，受限于该部件的执行时间占系统中总执行时间的百分比。

系统性能加速比：

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

- 加速比依赖于两个因素
  - 可改进比例（F_e）：在改进前的系统中，可改进部分的执行时间在总的执行时间中所占的比例。

    它总是小于等于1。

    例如：一个需运行60秒的程序中有20秒的运算可以加速，那么这个比例就是20/60。
  - 部件加速比（S_e）：可改进部分改进以后性能提高的倍数。它是改进前所需的执行时间与改进后执行时间的比。

    一般情况下部件加速比是大于1的。

    例如：若系统改进后，可改进部分的执行时间是2秒，而改进前其执行时间为5秒，则部件加速比为5/2。

- 改进后程序的总执行时间T_n

$$T_n = T_0\left(1 - Fe + \frac{Fe}{Se}\right)$$

  - T_0：改进前整个程序的执行时间
  - $1-F_e$：不可改进比例

系统加速比S_n为改进前与改进后总执行时间之比：

$$S_n = \frac{T_0}{T_n} = \frac{1}{(1 - Fe) + \frac{Fe}{Se}}$$

例1.1 将计算机系统中某一功能的处理速度加快15倍，但该功能的处理时间仅占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？

解 由题可知： $F_e = 40\% = 0.4$

$S_e = 15$

根据Amdahl定律可知：

$$S_n = \frac{1}{(1-Fe)+\frac{Fe}{Se}} = \frac{1}{(1-0.4)+\frac{0.4}{15}} \approx 1.6$$

采用此增强功能方法后，能使整个系统的性能提高到原来的1.6倍。

例1.2 某计算机系统采用浮点运算部件后，使浮点运算速度提高到原来的25倍，而系统运行某一程序的整体性能提高到原来的4倍，试计算该程序中浮点操作所占的比例。

解 由题可知： $S_e = 25$ $S_n = 4$

根据Amdahl定律可知：

$$4 = \frac{1}{(1 - Fe) + \frac{Fe}{25}}$$

由此可得：Fe = 78.1%

即程序中浮点操作所占的比例为78.1%。

- Amdahl定律：一种性能改进的递减规则
  - 如果仅仅对计算任务中的一部分做性能改进，则改进得越多，所得到的总体性能的提升就越有限。
- 重要推论：如果只针对整个任务的一部分进行改进和优化，那么所获得的加速比不超过：

  1/（1－可改进比例）

3. CPU性能公式

- 执行一个程序所需的CPU时间

CPU时间 = 执行程序所需的时钟周期数×时钟周期时间

其中：时钟周期时间是系统时钟频率的倒数。

- 每条指令执行的平均时钟周期数CPI

（Cycles Per Instruction）

CPI = 执行程序所需的时钟周期数 / IC

IC：所执行的指令条数

- 程序执行的CPU时间可以写成

CPU时间 = IC ×CPI ×时钟周期时间

- CPU的性能取决于三个参数
  - 时钟周期时间：取决于硬件实现技术和计算机组成；
  - CPI：取决于计算机组成和指令系统的结构；
  - IC：取决于指令系统的结构和编译技术。
- 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：

$$\text{CPU时钟周期数} = \sum_{i=1}^{n} (CPI_i \times IC_i)$$

CPU时间 = 执行程序所需的时钟周期数×时钟周期时间

$$= \sum_{i=1}^{n} (CPI_i \times IC_i) \times 时钟周期时间$$

CPI可以表示为：

$$CPI = \frac{时钟周期数}{IC} = \frac{\sum_{i=1}^{n} (CPI_i \times IC_i)}{IC} = \sum_{i=1}^{n} (CPI_i \times \frac{IC_i}{IC})$$

其中：(IC_i/IC)反映了第i种指令在程序中所占的比例。

例1.3 假设FP指令的比例为25%，其中，FPSQR占全部指令的比例为2%，FP操作的CPI为4，FPSQR操作的CPI为20，其他指令的平均CPI为1.33。现有两种改进方案，第一种是把FPSQR操作的CPI减至2，第二种是把所有的FP操作的CPI减至2，试比较两种方案对系统性能的提高程度。

解 没有改进之前，每条指令的平均时钟周期CPI为：

$$CPI = \sum_{i=1}^{n}\left(CPI_i \times \frac{IC_i}{IC}\right) = (4 \times 25\%) + (1.33 \times 75\%) = 2$$

（1）采用第一种方案

FPSQR操作的CPI由$CPI_{FPSQR}=20$减至$CPI'_{FPSQR}=2$，则整个系统的指令平均时钟周期数为：

$$CPI_1=CPI—（CPI_{FPSQR}—CPI'_{FPSQR}）\times 2\%$$
$$=2—(20—2)\times 2\%=1.64$$

（2）采用第二种方案

所有FP操作的CPI由$CPI_{FP}=4$减至$CPI'_{FP}=2$，则整个系统的指令平均时钟周期数为：

$$CPI_2=CPI—（CPI_{FP}—CPI'_{FP}）\times 25\%$$
$$=2—(4—2)\times 25\%=1.5$$

从降低整个系统的指令平均时钟周期数的程度来看，第二种方案优于第一种方案。

例1.4 考虑条件分支指令的两种不同设计方法：

（1）CPU_1：通过比较指令设置条件码，然后测试条件码进行分支。

（2）CPU_2：在分支指令中包括比较过程。

在这两种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_1，执行的指令中分支指令占30%；由于每条分支指令之前都需要有比较指令，因此比较指令也占30%。由于CPU_1在分支时不需要比较，因此CPU_2的时钟周期时间是CPU_1的1.35倍。问：哪一个CPU更快？如果CPU_2的时钟周期时间只是CPU_1的1.15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30%，剩下的指令占用1个时钟周期。所以

$$CPI_1 = 0.3 \times 2 + 0.70 \times 1 = 1.3$$

则CPU_1性能为：

$$总CPU时间_1 = IC_1 \times 1.3 \times 时钟周期_1$$

根据假设，有：

$$时钟周期_2 = 1.35 \times 时钟周期_1$$

在CPU_2中没有独立的比较指令，所以CPU_2的程序量为CPU_1的70%，分支指令的比例为：

30%/70% = 42.8%

这些分支指令占用2个时钟周期，而剩下的57.2%的指令占用1个时钟周期，因此：

$$CPI_2 = 0.428 \times 2 + 0.572 \times 1 = 1.428$$

因为CPU_2不执行比较，故：

$$IC_2 = 0.8 \times IC_1$$

因此CPU_2性能为：

$$\text{总CPU时间}_2 = IC_2 \times CPI_2 \times \text{时钟周期}_2$$
$$= 0.7 \times IC_1 \times 1.428 \times (1.35 \times \text{时钟周期}_1)$$
$$= 1.349 \times IC_1 \times \text{时钟周期}_1$$

在这些假设之下，尽管CPU_2执行指令条数较少，CPU_1因为有着更短的时钟周期，所以比CPU_2快。

如果CPU_2的时钟周期时间仅仅是CPU_1的1.15倍，则

$$时钟周期_2 = 1.15 \times 时钟周期_1$$

CPU_2的性能为：

$$总CPU时间_2 = IC_2 \times CPI_2 \times 时钟周期_2$$
$$= 0.7 \times IC_1 \times 1.428 \times (1.15 \times 时钟周期_1)$$
$$= 1.15 \times IC_1 \times 时钟周期_1$$

因此CPU_2由于执行更少指令条数，比CPU_1运行更快。

4. 程序的局部性原理

程序执行时所访问的存储器地址分布不是随机的，而是相对地簇聚。

- 常用的一个经验规则

  程序执行时间的90%都是在执行程序中10%的代码。

- 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。

1.2.2 计算机系统设计者的主要任务

1. 计算机系统设计者的任务包括：指令系统的设计、数据表示的设计、功能的组织、逻辑设计以及其物理实现等。
2. 设计一个计算机系统大致要完成3个方面的工作。
   - 确定用户对计算机系统的功能、价格和性能的要求
     - 计算机系统设计者的目标

       设计出能满足用户的功能需求、有较长的生命周期、且又具有很高的性能价格比的系统。
     - 功能需求：根据市场的需要以及所设计系统的应用领域来确定

- 应用领域

  专用还是通用？面向科学计算还是面向商用处理？

- 软件兼容

  软件兼容是指一台计算机上的程序不加修改就可以搬到另一台计算机上正常运行。

- 操作系统需求

  包括地址空间大小、存储管理、保护等。从系统结构上对操作系统的需求提供支持，是很重要的一点。

- 标准

  确定系统中哪些方面要采用标准以及采用什么标准。

  如：浮点数标准、I/O总线标准、网络标准、程序设计语言标准等。

- 软硬件功能分配
  - 考虑如何优化设计？

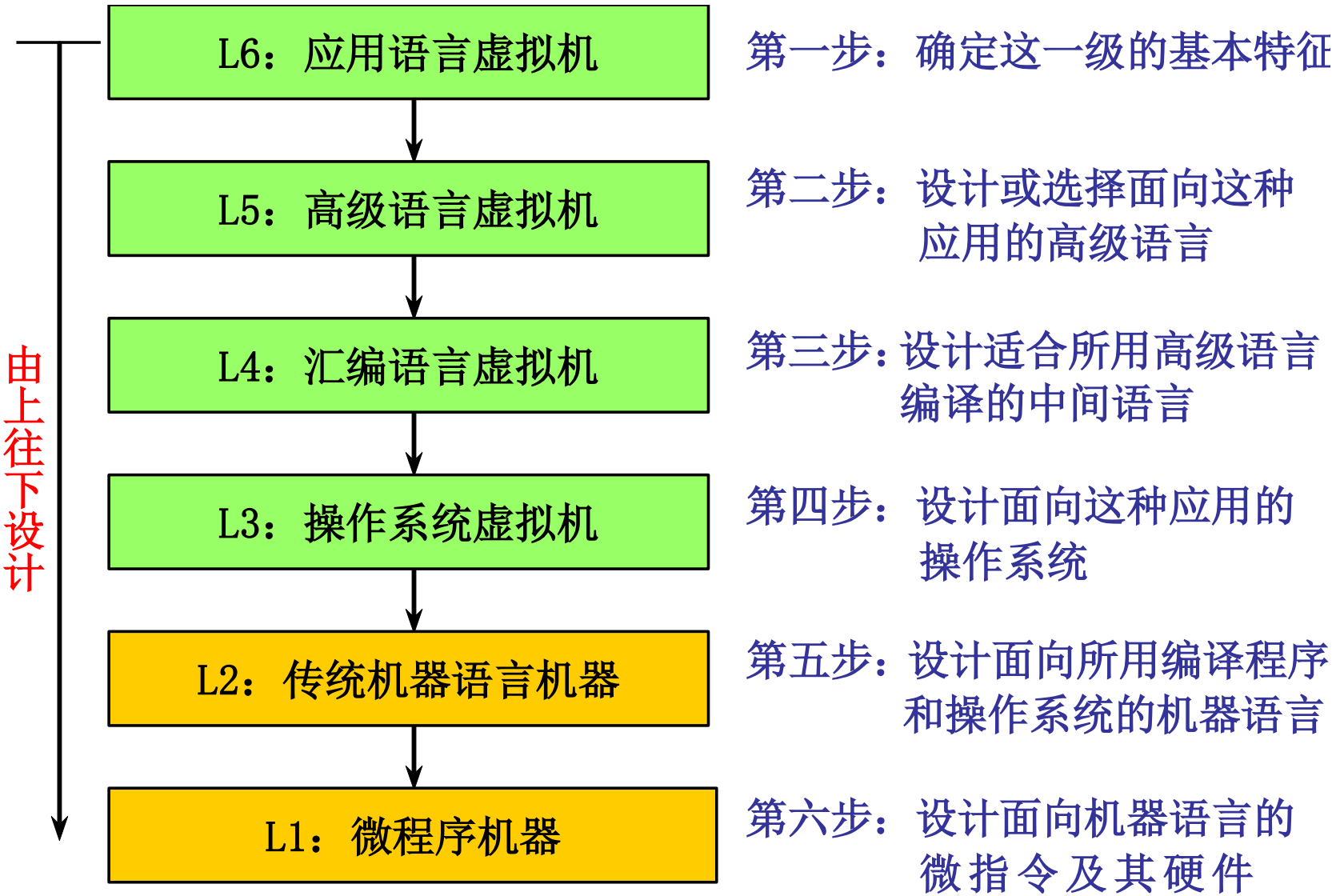
    必须考虑软硬件功能的合理分配。
  - 软件和硬件在实现功能上是等价的
    - 用软件实现的优点：设计容易、修改简单，而且可以减少硬件成本。但是所实现的功能的速度较慢。
    - 用硬件实现的优点：速度快、性能高，但它修改困难，灵活性差。
  - 在软硬件之间进行折中和取舍。

- 设计出生命周期长的系统结构
  - 特别注意计算机应用和计算机技术的发展趋势
  - 设计出具有一定前瞻性的系统结构，以使得它具有较长的生命周期。

1.2.3 计算机系统设计的主要方法

1. “由上往下”（top-down）设计

- 从层次结构中的最上面一级开始，逐层往下设计各层的机器。


L6：应用语言虚拟机
L5：高级语言虚拟机
L4：汇编语言虚拟机
L3：操作系统虚拟机
L2：传统机器语言机器
L1：微程序机器
由上往下设计
第一步：确定这一级的基本特征
第二步：设计或选择面向这种应用的高级语言
第三步：设计适合所用高级语言编译的中间语言
第四步：设计面向这种应用的操作系统
第五步：设计面向所用编译程序和操作系统的机器语言
第六步：设计面向机器语言的微指令及其硬件

- 首先确定面对使用者的那级机器的基本特征、数据类型和格式、基本命令等。
- 然后再逐级往下设计，每级都考虑如何优化上一级的实现。

➢ 适合于专用机的设计，而不适合通用机的设计。

2. “由下往上”（bottom-up）设计

➢ 从层次结构的最下面一级开始，逐层往上设计各层的机器。

➢ 采用这种方法时，软件技术完全处于被动状态，这会造成软件和硬件的脱节，使整个系统的效率降低。

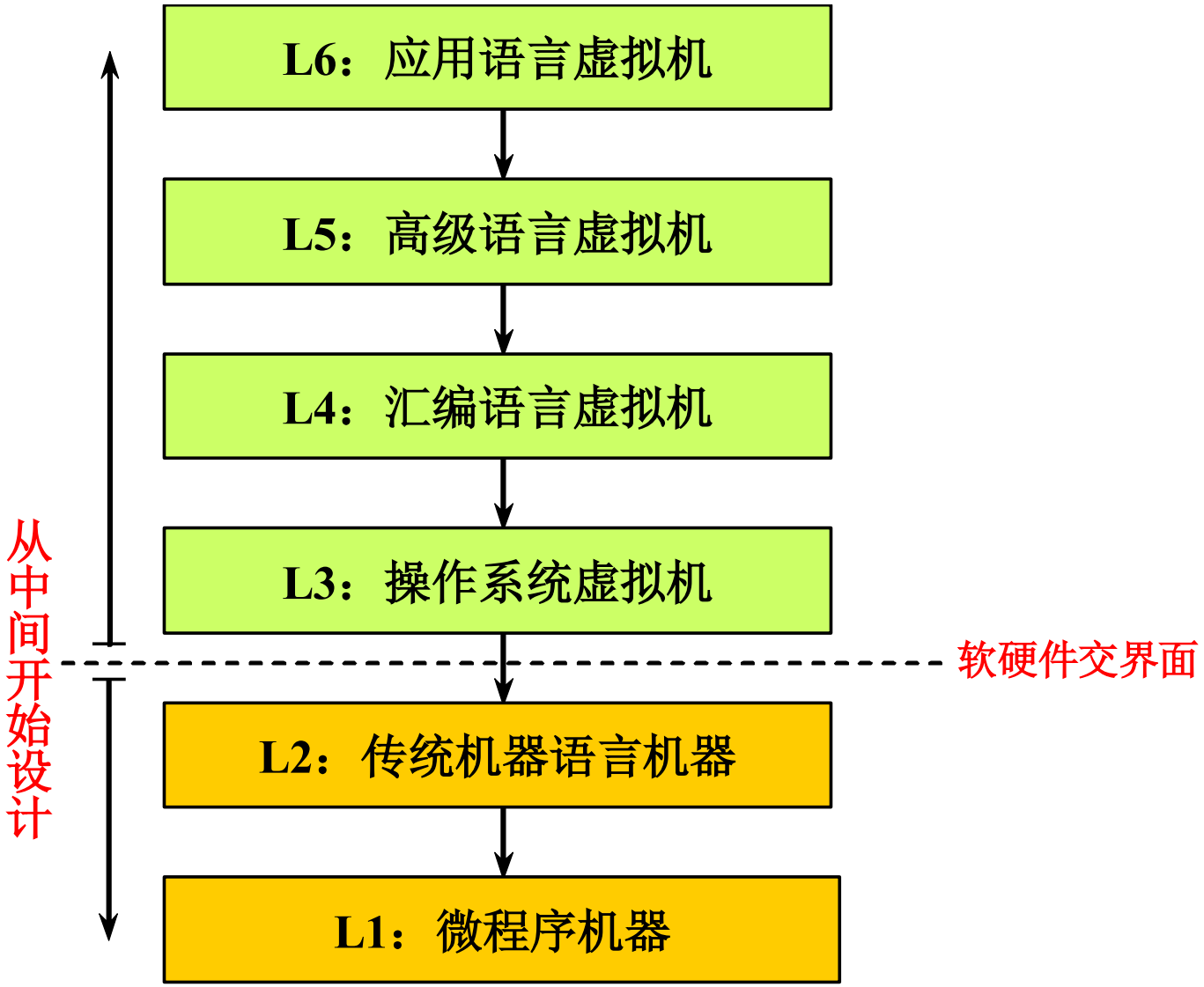
（在早期被采用得比较多，现在已经很少被采用了）

3. “从中间开始”（middle-out）设计

- “由上往下”和“由下往上”设计方法的主要缺点

  软、硬件设计分离和脱节

- 解决方法：综合考虑软、硬件的分工，从中间开始设计。
  - “中间”：层次结构中的软硬件的交界面，目前一般是在传统机器语言机器级与操作系统机器级之间。
- 从中间开始设计
  - 首先要进行软、硬件功能分配，确定好这个界面。
  - 然后从这个界面开始，软件设计者开始往上设计


L6：应用语言虚拟机
L5：高级语言虚拟机
L4：汇编语言虚拟机
L3：操作系统虚拟机
软硬件交界面
L2：传统机器语言机器
L1：微程序机器
从中间开始设计

操作系统、汇编、编译系统等，硬件设计者开始往下设计传统机器级、微程序机器级等。

1.3 计算机系统的性能评测

1. 执行时间和吞吐率

如何评测一台计算机的性能，与测试者看问题的角度有关。

- 用户关心的是：单个程序的执行时间（执行单个程序所花的时间很少）
- 数据处理中心的管理员关心的是：吞吐率（在单位时间里能够完成的任务很多 ）

假设两台计算机为X和Y，X比Y快的意思是：

对于给定任务，X的执行时间比Y的执行时间少。

X的性能是Y的n倍：

$$\frac{\text{执行时间Y}}{\text{执行时间X}}=n$$

执行时间与性能成反比：

$$n=\frac{\text{执行时间Y}}{\text{执行时间X}}=\frac{\frac{1}{\text{性能Y}}}{\frac{1}{\text{性能X}}}=\frac{\text{性能X}}{\text{性能Y}}$$

- 执行时间可以有多种定义：
  - 计算机完成某一任务所花费的全部时间，包括磁盘访问、存储器访问、输入/输出、操作系统开销等。
  - **CPU时间**：CPU执行所给定的程序所花费的时间，不包含I/O等待时间以及运行其它程序的时间。
    - 用户CPU时间：用户程序所耗费的CPU时间。
    - 系统CPU时间：用户程序运行期间操作系统耗费的CPU时间。

2. 基准测试程序

- 用于测试和比较性能的基准测试程序的最佳选择是真实应用程序。

  （例如编译器）

- 以前常采用简化了的程序，例如：
  - 核心测试程序：从真实程序中选出的关键代码段构成的小程序。
  - 小测试程序：简单的只有几十行的小程序。
  - 合成的测试程序：人工合成出来的程序。

  Whetstone与Dhrystone是最流行的合成测试程序。

从测试性能的角度来看，上述测试程序不可信了。

原因：

- 这些程序比较小，具有片面性；
- 系统结构设计者和编译器的设计者可以“合谋”把他们的机器面向这些测试程序进行优化设计，使得该机器显得性能更高。

- 性能测试的结果除了和采用什么测试程序有关以外，还和在什么条件下进行测试有关。
- 基准测试程序设计者对制造商的要求
  - 采用同一种编译器；
  - 对同一种语言的程序都采用相同的一组编译标志。

- 一个问题：是否允许修改测试程序的源程序

  三种不同的处理方法：

  - 不允许修改；
  - 允许修改，但因测试程序很复杂或者很大，几乎是无法修改。
  - 允许修改，只要保证最后输出的结果相同。

- 基准测试程序套件：由各种不同的真实应用程序构成。

  （能比较全面地反映计算机在各个方面的处理性能 ）

- SPEC系列：最成功和最常见的测试程序套件

  （美国的标准性能测试公司创建）

- 桌面计算机的基准测试程序套件可以分为两大类：
  处理器性能测试程序，图形性能测试程序
- SPEC89：用于测试处理器性能。10个程序（4个整数程序，6个浮点程序）
- 演化出了4个版本
  SPEC92：20个程序
  SPEC95：18个程序
  SPEC2000：26个程序
  SPEC CPU2006： 29个程序
- SPEC CPU2006

整数程序12个（CINT2006）

9个是用C写的，3个是用C++写的

浮点程序17个（CFP2006）

6个是用FORTRAN写的，4个是用C++写的，3个是用C写的，4个是用C和FORTRAN混合编写的。

- SPEC测试程序套件中的其它一系列测试程序组件
  - SPECSFS：用于NFS（网络文件系统）文件服务器的测试程序。它不仅测试处理器的性能，而且测试I/O系统的性能。它重点测试吞吐率。
  - SPECWeb：Web服务器测试程序。

- SPECviewperf：用于测试图形系统支持OpenGL库的性能。
- SPECapc：用于测试图形密集型应用的性能。

3. 性能比较

两个程序在A、B、C三台机器上的执行时间

	机器A	机器B	机器C	W（1）	W（2）	W（3）
程序1	1.00	10.00	20.00	0.50	0.909	0.999
程序2	1000.00	10.00	20.00	0.50	0.091	0.001
加权算术平均值A_m（1）	500.50	10.00	20.00			
加权算术平均值A_m（2）	91.91	10.00	20.00			
加权算术平均值A_m（3）	2.00	10.00	20.00			

如何比较这三台机器的性能呢？

从该表可以得出：

执行程序1：

- A机的速度是B机的10倍
- A机的速度是C机的20倍
- B机的速度是C机的2倍

执行程序2：

- B机的速度是A机的100倍
- C机的速度是A机的50倍
- B机的速度是C机的2倍

- **总执行时间**：机器执行所有测试程序的总时间
  - B机执行程序1和程序2的速度是A机的50.05倍
  - C机执行程序1和程序2的速度是A机的24.02倍
  - B机执行程序1和程序2的速度是C机的2倍
- **平均执行时间**：各测试程序执行时间的算术平均值

$$S_m = \frac{1}{n}\sum_{i=1}^{n} T_i$$

其中：T_i：第i个测试程序的执行时间

n：测试程序组中程序的个数

- **加权执行时间**：各测试程序执行时间的加权平均值

$$A_m = \sum_{i=1}^{n} W_i \cdot T_i$$

其中，W_i：第i个测试程序在测试程序组中所占的比重

$$\sum_{i=1}^{n} W_i = 1$$

T_i：该程序的执行时间

➢ 调和平均值法

$$H_m = \frac{n}{\sum_{i=1}^{n}\frac{1}{R_i}} = \frac{n}{\sum_{i=1}^{n}T_i}$$

其中，R_i：由n个程序组成的工作负荷中执行第i个程序的速度

$R_i = 1/T_i$

T_i：第i个程序的执行时间

- 加权调和平均值公式

$$H_m = \left(\sum_{i=1}^{n}\frac{W_i}{R_i}\right)^{-1} = \left(\sum_{i=1}^{n}W_iT_i\right)^{-1}$$

- **几何平均值法**：以某台计算机的性能作为参考标准，其他计算机性能则除以该参考标准而获得一个比值。

$$G_m = \sqrt[n]{\prod_{i=1}^{n} R_i} = \sqrt[n]{\prod_{i=1}^{n} \frac{1}{T_i}}$$

R_i：由n个程序组成的工作负荷中执行第i个程序的速度

$R_i = 1/T_i$

$\prod$：连乘

- 加权几何平均值

$$G_m = \prod_{i=1}^{n} (R_i)^{W_i} = (R_1)^{W_1} \times (R_2)^{W_2} \times \cdots \times (R_n)^{W_n}$$

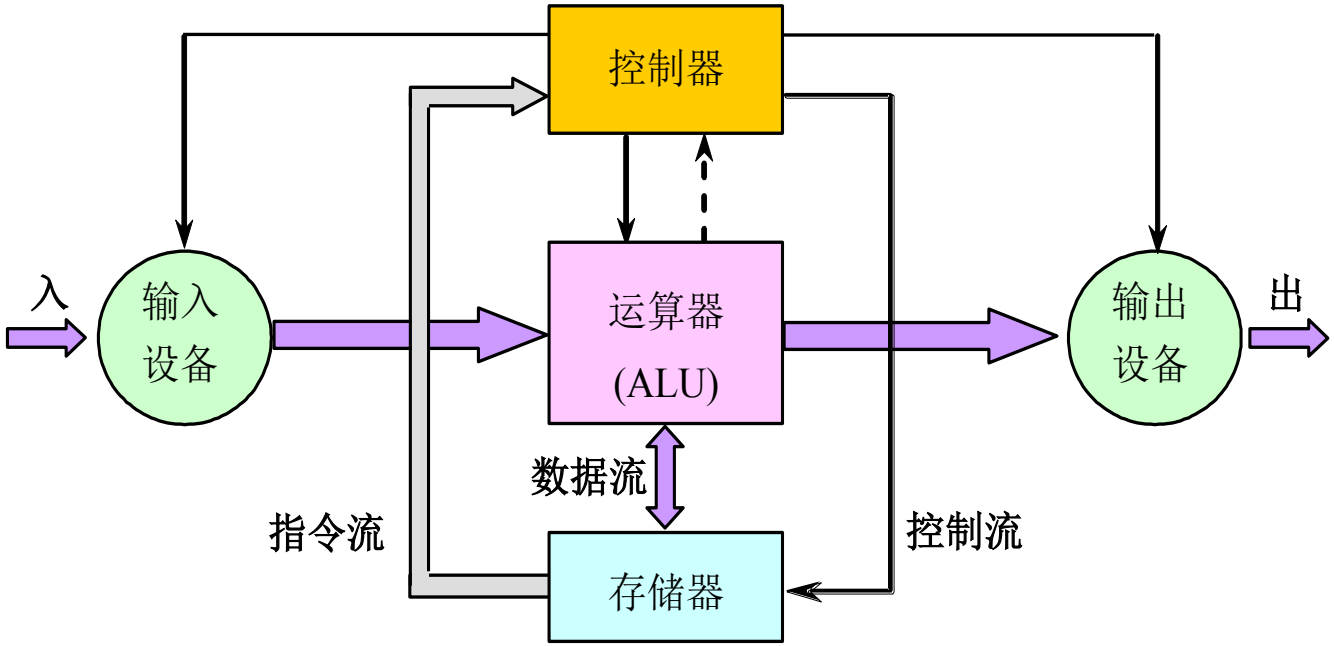
- G_m表示法有一个很好的特性

  几何平均值的比等于比的几何平均值

$$\frac{G_m(x_i)}{G_m(y_i)} = G_m\left(\frac{x_i}{y_i}\right)$$

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

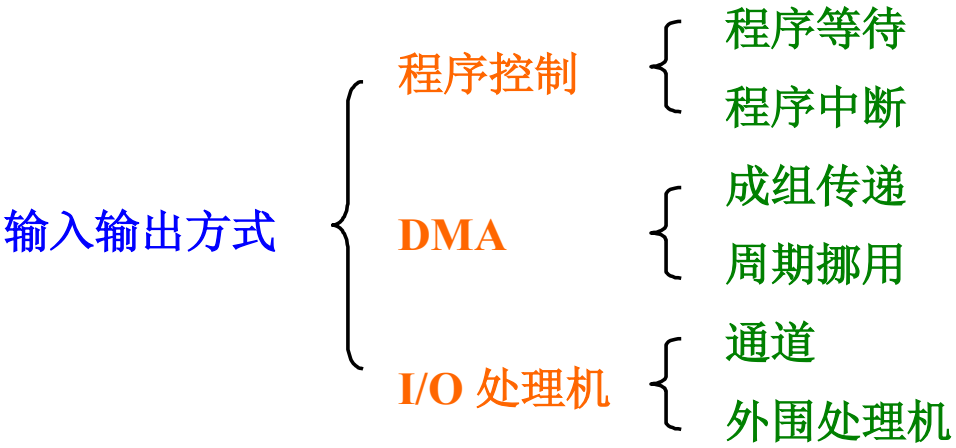
- 计算机以运算器为中心。
- 在存储器中，指令和数据同等对待。

  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

1.4.2 软件对系统结构的影响

- 软件的可移植性：一个软件可以不经修改或者只需少量修改就可以由一台机器移植到另一台机器上正确地运行。差别只是执行时间的不同。

  我们称这两台机器是软件兼容的。

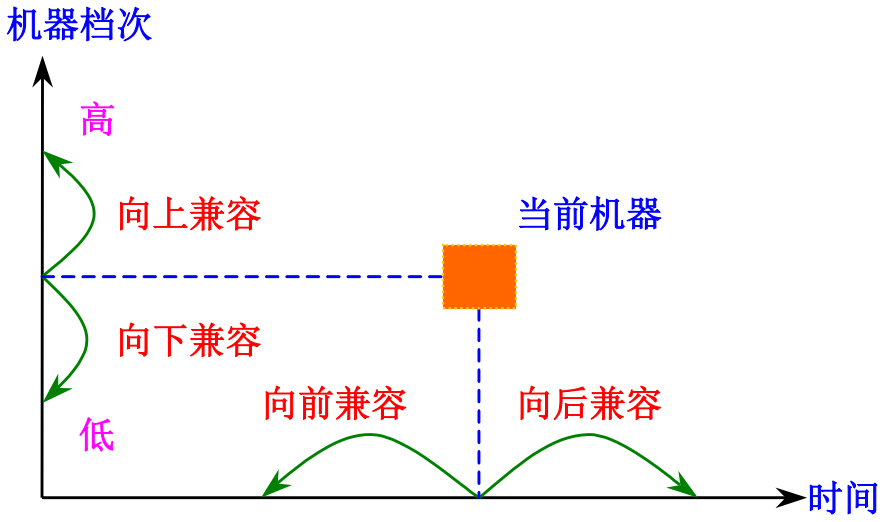
- 实现可移植性的常用方法

  采用系列机，模拟与仿真，统一高级语言 。

1. 统一高级语言
   - 实现软件移植的一种理想的方法
   - 较难实现
2. 系列机

   由同一厂家生产的具有相同的系统结构，但具有不同组成和实现的一系列不同型号的机器。
   - 较好地解决软件开发要求系统结构相对稳定与器件、硬件技术迅速发展的矛盾。
   - 软件兼容

- **向上（下）兼容：**按某档机器编制的程序，不加修改就能运行于比它高（低）档的机器。
- **向前（后）兼容：**按某个时期投入市场的某种型号机器编制的程序，不加修改地就能运行于在它之前（后）投入市场的机器。



- 向后兼容是系列机的根本特征。

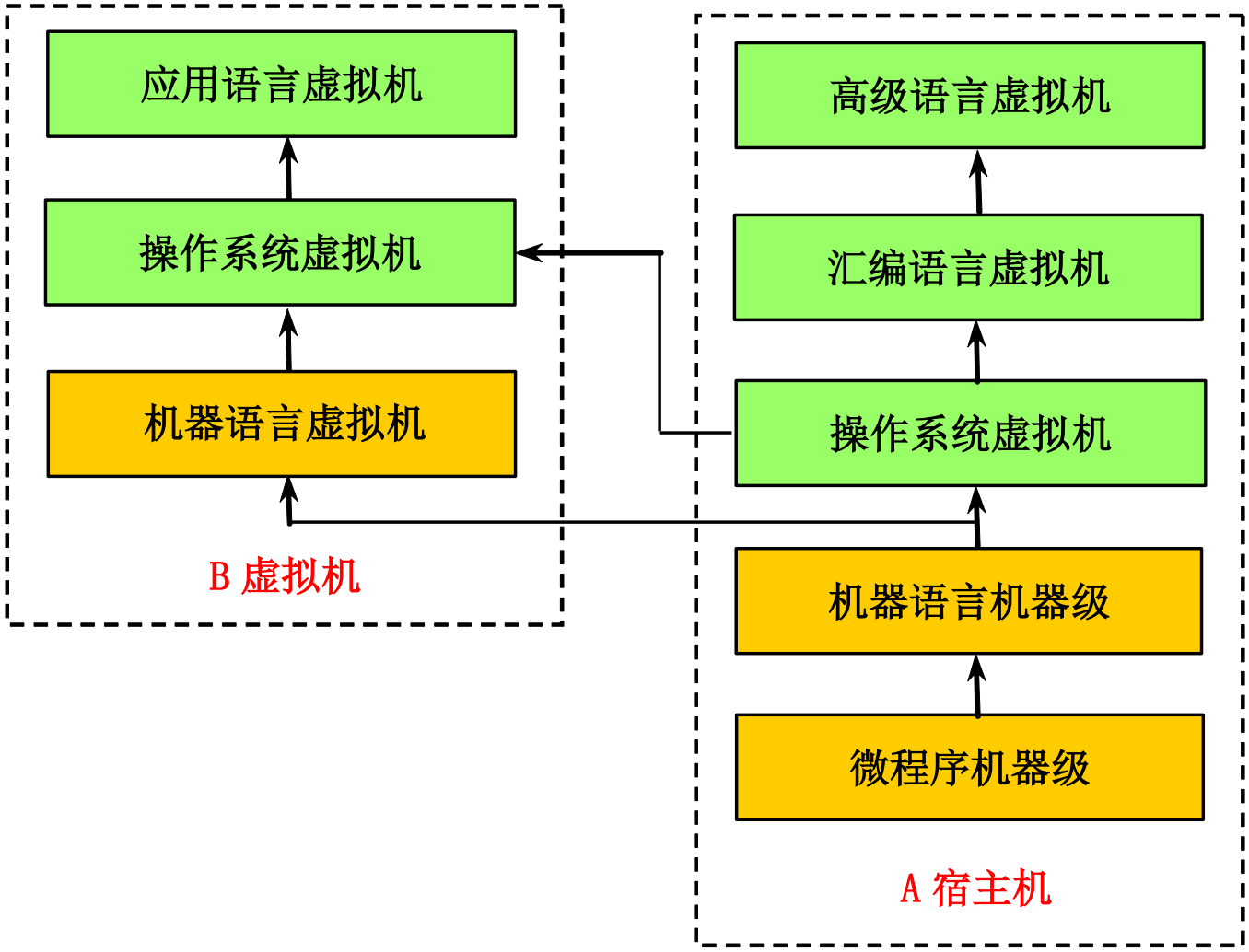
➢ 兼容机：由不同公司厂家生产的具有相同系统结构的计算机 。

3. 模拟和仿真

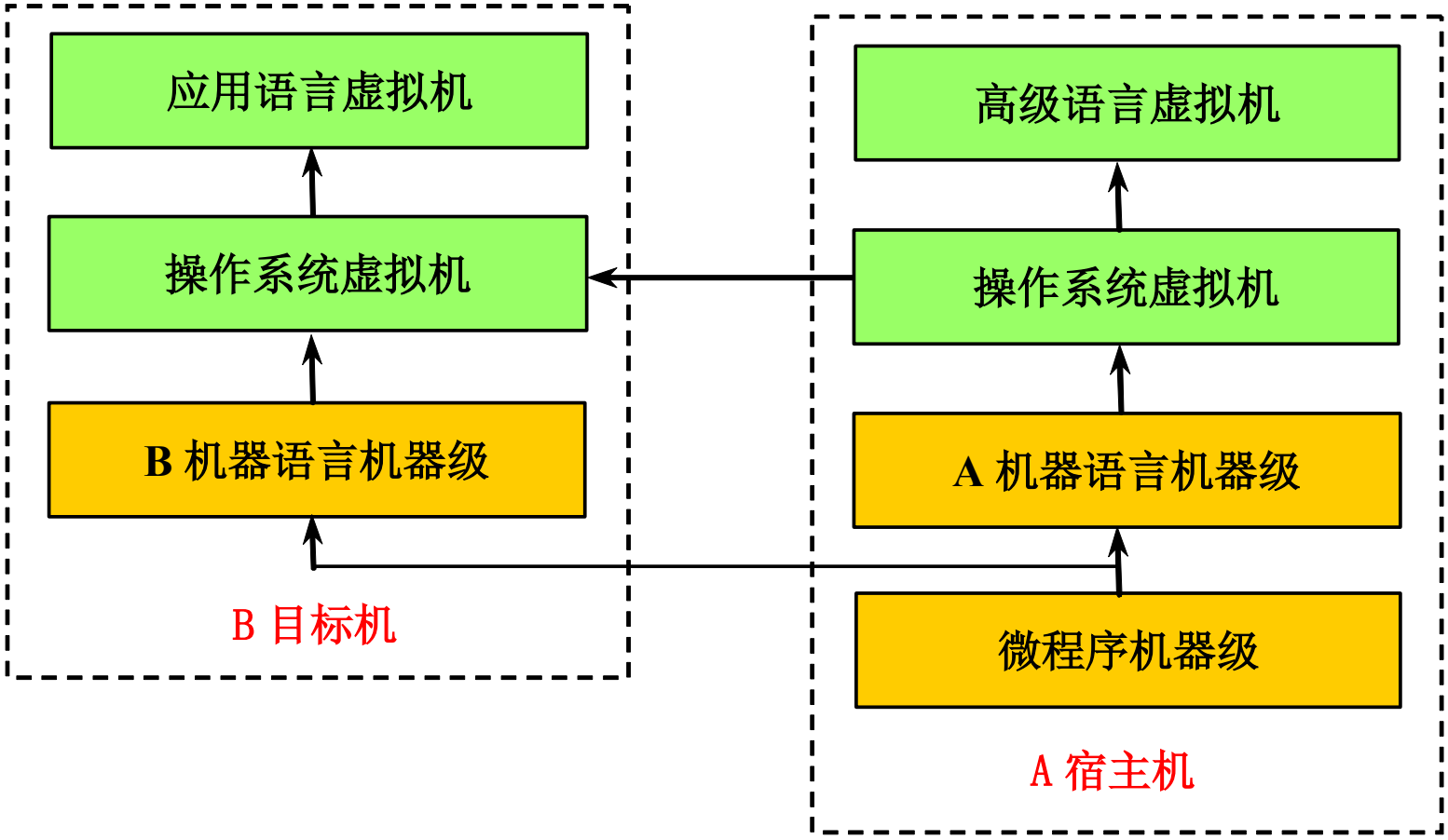
➢ 使软件能在具有不同系统结构的机器之间相互移植。

- 在一种系统结构上实现另一种系统结构。
- 从指令集的角度来看，就是要在一种机器上实现另一种机器的指令集。

- 模拟：用软件的方法在一台现有的机器（称为宿主机）上实现另一台机器（称为虚拟机）的指令集。
  - 通常用解释的方法来实现。
  - 运行速度较慢，性能较差。


应用语言虚拟机
操作系统虚拟机
机器语言虚拟机
B 虚拟机
高级语言虚拟机
汇编语言虚拟机
操作系统虚拟机
机器语言机器级
微程序机器级
A 宿主机

- 仿真：用一台现有机器（宿主机）上的微程序去解释实现另一台机器（目标机）的指令集。
  - 运行速度比模拟方法的快
  - 仿真只能在系统结构差距不大的机器之间使用


应用语言虚拟机
操作系统虚拟机
B 机器语言机器级
B 目标机
高级语言虚拟机
操作系统虚拟机
A 机器语言机器级
微程序机器级
A 宿主机

1.4.3 器件发展对系统结构的影响

1. 推动计算机系统结构不断发展的最活跃的因素
2. 摩尔定律

   集成电路芯片上所集成的晶体管数目每隔18个月就翻一番。
3. 计算机的分代主要以器件作为划分标准。
   - 它们在器件、系统结构和软件技术等方面都有各自的特征。
     - **SMP**：对称式共享存储器多处理机

       MPP：大规模并行处理机　　**MP**：多处理机

分代	器件特征	结构特征	软件特征	典型实例
第一代 （1945—1954年）	电子管和继电器	存储程序计算机 程序控制I/O	机器语言 汇编语言	普林斯顿ISA， ENIAC，IBM 701
第二代 （1955—1964年）	晶体管、磁芯 印刷电路	浮点数据表示 寻址技术 中断、I/O处理机	高级语言和编译 批处理监控系统	Univac LAPC， CDC 1604， IBM 7030
第三代 （1965—1974年）	SSI和MSI 多层印刷电路 微程序	流水线、Cache 先行处理 系列机	多道程序 分时操作系统	IBM 360/370， CDC 6600/7600， DEC PDP-8
第四代 （1975—1990年）	LSI和VLSI 半导体存储器	向量处理 分布式存储器	并行与分布处理	Cray-1，IBM 3090，DEC VAX 9000，Convax-1
第五代 （1991年—）	高性能微处理器 高密度电路	超标量、超流水 SMP、MP、MPP 机群	大规模、可扩展 并行与分布处理	SGI Cray T3E， IBM SP2，DEC AlphaServer 8400

1.4.4 应用对系统结构的影响

1. 不同的应用对计算机系统结构的设计提出了不同的要求。
2. 应用需求是促使计算机系统结构发展的最根本的动力。
3. 一些特殊领域：需要高性能的系统结构
   - 高结构化的数值计算

     气象模型、流体动力学、有限元分析
   - 非结构化的数值计算

     蒙特卡洛模拟、稀疏矩阵
   - 实时多因素问题

     语音识别、图象处理、计算机视觉

- 大存储容量和输入输出密集的问题

  数据库系统、事务处理系统

- 图形学和设计问题

  计算机辅助设计

- 人工智能

  面向知识的系统、推理系统等

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：

1. 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。

2. 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。

3. 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

1. 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。

2. 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 。

3. 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。

1.5.5 并行机的发展变化

并行机的发展可分为4个阶段。

1. 并行机的萌芽阶段（1964年～1975年）

- 20世纪60年代初期
  - **CDC6600**：非对称的共享存储结构，中央处理机采用了双**CPU**，并连接了多个外部处理器。
- 60年代后期，一个重要的突破
  - 在处理器中使用流水线和重复设置功能单元，所获得的性能提高是明显的，并比单纯地提高时钟频率来提高性能更有效。

- 在1972年，Illinois大学和Burroughs公司联合研制Illiac Ⅳ SIMD计算机（64个处理单元构成的）

  在1975年 Illiac Ⅳ系统 （16个处理单元构成）

2. 向量机的发展和鼎盛阶段（1976年～1990年）
   - 1976年，Cray公司推出了第一台向量计算机Cray-1
   - 在随后的10年中，不断地推出新的向量计算机。

     包括：CDC的Cyber205、Fujitsu的VP1000/VP2000、NEC的SX1/SX2以、我国的YH-1等
   - 向量计算机的发展呈两大趋势
     - **提高单处理器的速度**
     - **研制多处理器系统**

3. MPP出现和蓬勃发展阶段（1990年～1995年）

- 早期的MPP
  - TC2000（1989年）、Touchstone Delta、Intel Paragon（1992年）、KSR1、Cray T3D（1993年）、IBM SP2（1994年）和我国的曙光-1000（1995年）等。（分布存储的MIMD计算机）
- MPP的高端机器
  - 1996年，Intel公司的ASCI Red和1997年SGI Cray公司的T3E900 （万亿次高性能并行计算机）
- 90年代的中期，在中、低档市场上，SMP以其更优的性能/价格比代替了MPP。

4. 各种体系结构并存阶段（1995年～2000年）

- 从1995年以后，PVP（并行向量处理机）、MPP、SMP、DSM（分布式共享存储多处理机）、COW等各种体系结构进入并存发展的阶段。
- MPP系统在全世界前500强最快的计算机中的占有量继续稳固上升，其性能也得到了进一步的提高。

  如：**ASCI Red**的理论峰值速度已达到了**1Tflop/s**

  SX4和**VPP700**等的理论峰值速度也都达到了**1Tflop/s**

5. 机群蓬勃发展阶段（2000年以后）

- **机群系统**：将一群工作站或高档微机用某种结构的互连网络互连起来，充分利用其中各计算机的资源，统一调度、协调处理，以达到很高的峰值性能，并实现高效的并行计算。
- 1997年6月才有第一台机群结构的计算机进入Top500排名
- 2003年11月，这一数字已达到208台，机群首次成为Top500排名中比例最高的结构。
- 截至2008年6月，机群已经连续10期位居榜首，其数量已经达到400，占80%。

- 机群已成为当今构建高性能并行计算机系统的最常用的结构。
- 1993年至2008年期间，Top500中机群和MPP的数量的分布情况。