计算机体系结构复习

一、期末考试题型

1. 选择题，每小题 2 分，计 10 分；
2. 填空题，每空 1 分，计 10 分；
3. 简答题，每题 6 分，五题，计 30 分；
4. 综合题，共 6 题，计 50 分。

二、重点章节

> **重点章节的内容会放在 第三部分：知识点 里面**

1. 流水线（参考知识点 1、2、10）

2. 相关处理（参考知识点 4）

3. ILP 提升技术（参考知识点 9）

> ILP（Instruction-Level-Parallelism）指令级别并行

4. 存储层次架构（参考知识点 8）

5. 缓存（参考知识点 3、6、7）

6. 虚拟存储器（参考知识点 11）

三、知识点

1. Amdahl 定律

① Amdahl的定义

> **系统中对某一部件采用更快执行方式所能获得的系统性能改进程度，取决于这种执行方式被使用的频率，或所占总执行时间的比例。**阿姆达尔定律实际上定义了采取增强（加速）某部分功能处理的措施后可获得的性能改进或执行时间的加速比。简单来说是通过更快的处理器来获得加速是由慢的系统组件所限制。
>
> 实际上Amdahl定律是一种求加速比的方法。

② Amdahl公式

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

③ Amdahl观点

- **性能增加的递减规则**
  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
- **Amdahl定律的重要推论**
  针对整个任务的一部分进行优化，则最大加速比不大于

$$\frac{1}{1 - \text{可改进比例}}$$

- **衡量“好”的计算机系统**

  具有高性能价格比的计算机系统是一个带宽平的系统，而不是看它使用的某些部件的性能

---

2. 性能分析

① CPU 的性能

- 程序执行过程中所处理的指令数，记为**IC**。
- 每条指令执行所需要的时钟周期数 **CPI** (Cycles Per Instruction）。
- 每条指令执行所需要的平均时钟周期数

$$CPI = CLK/IC$$

② CPU 性能公式

> 假设计算机系统有n种指令，其其中第 i 种指令的处理时间为 Latex CPI_i，在程序中第 i 种指令出现的次数为 IC_i

$$\begin{aligned} T_{CPU} &= CLK/f \\ CLK &= \sum(IC_i \times CPI_i) \\ T_{CPU} &= \sum(IC_i \times CPI_i)/f \\ &= \sum(IC_i \times CPI_i) \times T_{CLK} \end{aligned}$$

---

3. 缓存性能优化

① 降低 Cache 失效率方法（8种）

（1）增加Cache块大小。增加块大小利用了程序的空间局部性。

（2）增加Cache的容量。

（3）提高相联度，降低冲突失效。

（4） 伪相联Cache，降低冲突失效。当对伪相联Cache进行访问时，首先是按与直接映象相同的方式进行访问。如果命中，则从相应的块中取出所访问的数据，送给 CPU，访问结束。如果不命中，就将索引字段的最高位取反，然后按照新索引去寻找“伪相联组”中的对应块。如果这一块的标识匹配，则称发生了“伪命中”。否则，就访问下一级存储器。

（5）硬件预取技术。在处理器提出访问请求前预取指令和数据。

（6）由编译器控制的预取。硬件预取的替代方法，在编译时加入预取的指令，在数据被用到之前发出预取请求。

（7）编译器优化。通过对软件的优化来降低失效率。

（8）“牺牲” Cache。在Cache和其下一级存储器的数据通路之间增设一个全相联的小Cache，存放因冲突而被替换出去的那些块。每当发生不命中时，在访问下一级存储器之前，先检查“牺牲”Cache中是否含有所需的块。如果有，就将该块与Cache中某个块做交换，把所需的块从“牺牲”Cache 调入Cache。

② 减少 Cache 失效开销方法（5种）

(1) 让读失效优先于写。

(2) 写缓冲合并。

(3) 请求字处理技术。

- 尽早重启动： 在请求字没有到达时，CPU 处于等待状态。一旦请求字到达， 就立即发送给 CPU，让等待的 CPU 尽早重启动，继续执行。
- 请求字优先： 调块时，首先向存储器请求 CPU 所要的请求字。请求字一旦到 达，就立刻送往 CPU，让 CPU 继续执行，同时从存储器调入该块的其余部、 分。请求字优先也称为回绕读取或关键字优先。

(4) 非阻塞 Cache 或非锁定 Cache 技术。

(5) 采用二级 Cache。

③ 减少 Cache 命中时间方法（4种）

容量小且结构简单的 Cache、虚拟 Cache 、 Trace Cache、 Cache 访问流水化。

---

4. 相关和冒险的分析

> 如果存在数据相关，硬件检测机制会做如下的事情：
>
> 知道相关消除动态调度暂停指令停止取指令和发射指令静态调度（开始于60s，流行于80s）消除动态调度软件来负责调度指令减少空转动态调度硬件对指令的执行重新排序来减少空转

① 相关(dependence)及其类型

- **在某些场合又叫做：“hazard” 冒险**
- 相关限定了指令执行时的顺序要求
- **相关类型**：
  - **数据相关**： Data dependence
    - 流相关 (真相关 – read after write， RAW)
    - 输出相关 (write after write， WAW)
    - 反相关 (write after read， WAR)

> **流相关**是必须一直要遵从的，因为构成了指令间基于值的依赖关系
>
> **反相关**和**输出相关**出席的原因，是由于体系结构中的硬件寄存器数量不足

  - **控制相关**： Control dependence
  - **资源相关**

② 相关检测的方法

（1）**计分板/记分牌方法**

- 为寄存器文件中每个寄存器配置有效位（valid bit）
- 写某个寄存器的指令重置（reset）相应的有效位
- 一个处于译码阶段的指令检查其源和目的寄存器的有效位是否为valid
  - 如是: 不需要暂停…无数据相关
  - 否则: 暂停指令的处理
- **优点**：简单，每个寄存器1比特开销
- **缺点**：所有数据相关都暂停指令处理，有点极端

（2）**基于组合逻辑的数据相关检测**

- 用特殊的检测逻辑来检测当前处于后续流水段中的指令是否会写当前处于译码阶段的指令的源寄存器
  - 如是: 暂停译码段中指令的处理
  - 否则: 没必要暂定流水线…没有真相关
- **优点**: 反相关和输出相关时，没必要暂停流水线
- **缺点**: 检测电路比记分牌算法中的逻辑开销大。对于深度流水线和超标量来说，开销更大

5.分支预测技术

① 动态分支预测

- **定义：**在程序运行时，根据分支指令过去的表现来预测其将来的行为。
  - 如果分支行为发生了变化，预测结果也跟着改变。
  - 相比于静态预测，有更好的预测准确度和适应性。
- **分支预测的有效性取决于**
  - 预测的准确性。
  - 预测正确和不正确时的开销。
- **决定分支开销的因素**
  - 流水线的结构。
  - 具体的预测方法。
  - 预测错误时的恢复策略等。

② 分支历史表 BHT

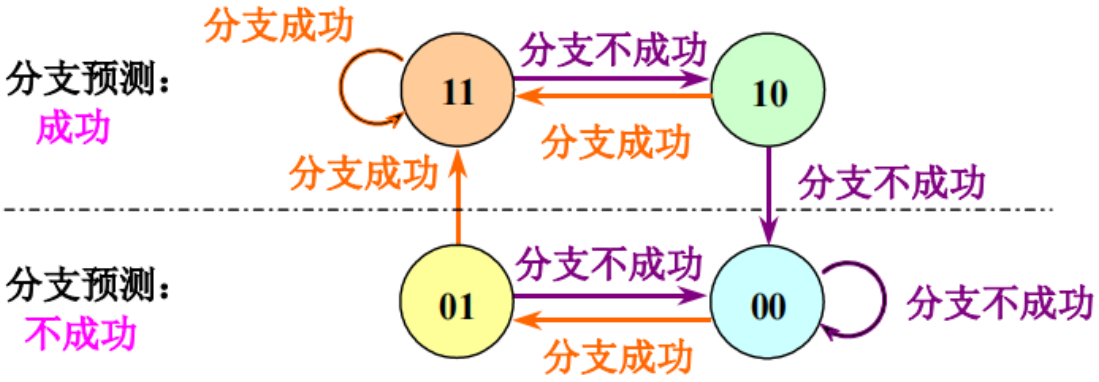
- **定义**：**BHT（Branch History Table）**，这是记录分支历史信息的表格，用于判定一条分支指令是否token，记录的是跳转信息。
- **简单方法**：用1bit位记录，例如1表示跳转，0表示不跳转，而这个表格的索引是指令PC值。

> 考虑在32位系统中，如果要记录完整32位的branch history，则需要4Gbit的存储器，这超出了系统提供的硬件支持能力；所以一般就用指令的后12位作为BHT表格的索引，这样用4Kbit的一个表格，就可以记录branch history了。

- **精确方法**：在BHT中用1bit位记录分支是否跳转还不够准确，可以使用2bit位记录（而用*3bit*或者更多位记录，效果与*2bit*类似）所以在BHT中，一般用2bit位记录分支是否跳转。这个2bit计数器大伙叫做饱和计数器。

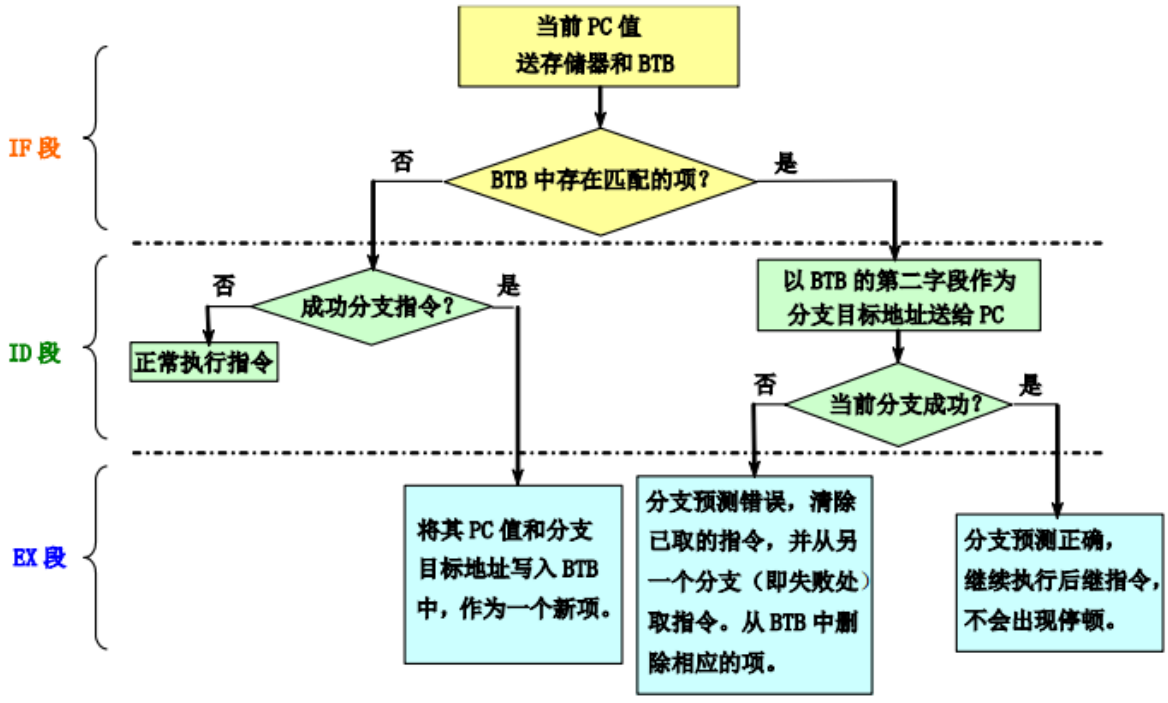
> 一般11和10表示这条分支会跳转；01和00表示分支不会跳转。

> 研究表明：对于SPEC89测试程序来说，具有大小为4K的BHT的预测准确率为82%～99%。一般来说，采用4K的BHT就可以了。



③ 分支目标缓冲器 BTB

- **定义**：BTB（Branch-Target Buffer）用于记录一条分支指令的跳转地址，将分支成功的分支指令的地址和它的分支目标地址都放到一个缓冲区中保存起来，缓冲区以分支指令的地址作为标识。由于这儿存储的是指令地址，例如32位地址，因此，这个表格就不能做到存储BHT那样多的内容了，如果也支持4K条指令，则需要128Kbit的存储空间，这几乎可以赶上一个L1Cache的容量了，所以BTB一般很小，就32项或者64项。由于这个BTB容量小，并且其用于是记录分支指令的跳转地址，因此，如果这条指令不跳转，即其下一条指令就是PC+4，则不会在BTB中记录的。



- **BTB操作延迟**

指令在BTB中?	预测	实际情况	延迟周期
是	成功	成功	0
是	成功	不成功	2
不是		成功	2
不是		不成功	0

- **基于BTB和BHT的分支预测**
  （1）在取指阶段利用PC寻址BTB，如果命中，则说明这是一条跳转指令，利用从BTB中获取到的地址去取icache；
  （2）由于BTB中保存的内容不够多，因此BHT的准确率更高，这个时候索引BHT表格，如果发现BHT也跳转，则说明这条指令预测是跳转的；如果BHT不跳转，则说明不跳转，这个时候就取消BTB中的指令地址，重新PC+4去取icache；

④ 前瞻(推测)执行：还没整

---

6. 缓存失效

① 强制性失效

当第一次访问一个块时，该块不在 Cache 中，需从下一级存储器中调入 Cache，这就是强制性失效。

② 容量失效

如果程序执行时所需的块不能全部调入 Cache 中，则当某些块被替换后，若又重新被访问，就会发生失效。这种失效称为容量失效。

③ 冲突失效

在组相联或直接映像 Cache 中，若太多的块映像到同一组（块）中，则会出现该组中某个块被别的块替换（即使别的组或块有空闲位置），然后又被重新访问的情况。这就发生了冲突失效。

---

7. 缓存替换方法

① 替换算法,各有什么优缺点?

（1）**随机法**：简单、易于用硬件实现，但这种方法没有考虑 Cache 块过去被使用的情况，反映不了程序的局部性，所以其失效率比 LRU 的高。

（2）**先进先出法**：容易实现。它虽然利用了同一组中各块进入 Cache 的顺序这一“历史”信息，但还是不能正确地反映程序的局部性。

（3）**最近最少使用法 LRU**：失效率最低。但是 LRU 比较复杂，硬件实现比较困难。

---

8. DRAM 组织（没整）

> PTT 主存 5.6.1

① 增加存储器宽度

② 采用简单的多体交叉存储器

③ 独立存储体

---

9. 乱序执行原理

- 乱序执行（out-of-order execution）是指 CPU 采用了允许将多条指令不按程序规定的顺序执行，并将执行结果分开发送给各相应电路单元处理的技术。比方 Core 乱序执行引擎说程序某一段有 7 条指令，此时 CPU 将根据各单元的空闲状态和各指令能否提前执行的具体情况分析后，将能提前执行的指令立即发送给相应电路执行。
- 在各单元不按规定顺序执行完指令后还必须由相应电路再将运算结果重新按原来程序指定的指令顺序排列后才能返回程序。这种将各条指令不按顺序拆散后执行的运行方式就叫乱序执行（也有叫错序执行）技术。
- 这样将根据个电路单元的状态和各指令能否提前执行的具体情况分析后，将能提前执行的指令立即发送给相应电路单元执行，在这期间不按规定顺序执行指令，然后由重新排列单元将各执行单元结果按指令顺序重新排列。采用乱序执行技术的目的是为了使 CPU 内部电路满负荷运转并相应提高了 CPU 的运行程序的速度。
  分枝技术：（branch）指令进行运算时需要等待结果，一般无条件分支只需要按指令顺序执行，而条件分枝必须根据处理后的结果，再决定是否按原先顺序进行。

处理器基本上会按照程序中书写的机器指令的顺序执行。按照书写顺序执行称为按序执行（In-Order）。按照书写顺序执行时，如果从内存读取数据的加载指令、除法运算指令等延迟（等待结果的时间）较长的指令后面紧跟着使用该指令结果的指令，就会陷入长时间的等待。尽管这种情况无可奈何，但有时，再下一条指令并不依赖于前面那条延迟较长的指令，只要有了操作数就能执行。此时可以打乱机器指令的顺序，就算指令位于后边，只要可以执行，就先执行，这就是乱序执行（Out-of-Order）。

> 乱序执行时，由于数据依赖性而无法立即执行的指令会被延后，因此可以减轻数据灾难的影响。

也可看此博客 https://blog.csdn.net/gjq_1988/article/details/39520729

---

10. 流水线特性和性能优化

> **流水线深度受限于流水线(锁存器)的延迟和额外开销**

① 流水线的特点

- 流水过程有多个**相联系的子过程**组成，每个过程称为流水线的**级、段**，一条流水线的段数，也叫做流水线的**深度或流水深度**
- 每个子过程由**专门的功能段**实现
- 每个功能段的**所需时间尽量相等**
- 流水线需要有**通过时间、排空时间**，只有经过通过时间后，流水线过程才能进入**稳定**的工作状态，流水线都不是满负荷工作
- 流水线技术适合**大量重复**的时序过程，只有在输入端能**连续**的提供任务时，它的效率才能充分发挥

② 流水线解决瓶颈的方法

- 细分瓶颈段
- 重复设置瓶颈段

③ 通过软件\编译器减少分支延迟方法（4种）

- **冻结或排空流水线**：一旦流水线的译码阶段检测到分支指令的话，就暂停后面的所有指令，直到分支指令**到达访存段之后，确定他成功并计算出新的 PC为止
- **预测分支失败**：我们沿着失败的分支继续处理指令，就好像什么都没有发生，当确定分支式失败时，说明预测正确，流水线正常流动
- **预测分支成功**：当确定分支式成功时，流水线按分支目标地址重新取指令执行
- **延迟分支**：主要思想是从逻辑上来延长分支指令的执行时间，把延迟分支看成是由原来的若干个延迟草和分支指令构成，不管分支是否成功都要按顺序执行延迟槽中的指令

> 后三种方法的共同特点是，**他们对于分支的处理方法在程序执行过程中始终是不变的**，他们总是要么预测分支成功，要么总是预测分支失败

④ 解决流水线数据冲突的方法（4种）

- **定向技术**：在某条指令产生一个结果之前，其他指令并不需要去真正的计算结果，如果将该结果从其产生的地方直接送到其他指令需要他的地方就可以避免暂停
- **暂停技术**：设置一个流水线互锁的功能部件，一旦流水线互锁检测到数据相关，流水线暂停执行，发生数据相关指令后续的所有指令，直到该数据相关解决为止
- **采用编译器调度**：为了减少停顿，对于无法用定向技术解决的数据冲突，可以通过在编译时让编译器实现指令调度或流水线调度来消除冲突
- **重新组织代码顺序**：重新组织指令顺序，以加大相关指令的距离，进而减少数据冲突的可能性

⑤ 流水线停顿的原因

- Stall（暂停）：导致流水线停止向后流动的一个条件
- 资源冲突
- 相关 (指令之间) **数据**或者**控制**
- 长延迟 (多周期) 操作

⑥ 流水线题目（没整完）

Q1：为什么流水线段数不是越多越好?

A1：流水线段数越多，锁存器越多，锁存器的延迟越大，同时锁存器的开销也越大。

---

11. 虚拟存储器原理和实现

① 虚拟存储器原理

- 由价格较贵、速度较快、容量较小的主存储器 M1 和一个价格低廉、速度较慢、容量很大的辅助存储器 M2 (通常是硬盘) 组成。在系统软件和辅助硬件的管理下，就象一个单一的、可直接访问的大容量主存储器。
- 程序员可以用机器指令的地址码对整个程序统一编址，就如同应用程序员具有对应于这个地址码宽度的存储空间 (称为程序空间) 一样，而不必考虑实际主存空间的大小。

② 虚存管理方式

> **分两类：页式和段式。**

- 页式虚存把空间划分为大小相同的块，称为页面。常用页大小为 4KB ~ 64KB 。
- 段式虚存把空间划分为可变长的块，称为段。段最小长度为 1 个字节，最大因机器而异，常为 2*16B ~ 2*32B 。

> 页面是对空间的机械划分，而段则往往是按程序的逻辑意义进行划分。

采用页式虚存\段式虚存特点

- 页式虚存：地址是单一、固定长度的地址字，由页号和页内位移两部分组成。
- 段式虚存：地址需要用两个字表示，一个为段号，另一个为段内位移。因为段的长度是可变的。
- 段页式：每段被划分成若干个页面。既保持了段作为逻辑单位的优点，又简化了替换的实现，而且段不必作为整体全部一次调入主存，而是可以以页面为单位部分调入。

除此之外，一级 cache 和虚拟存储器的区别：

cache 的块大小小于虚存，cache 命中时间小于虚存，失效开销小于虚存，但失效率大于虚存

③ 虚拟存储的四个问题

- **映象规则**： **全相联。**以降低失效率为主要目标。
- **查找算法**： **页表，段表， TLB 。**
  - **页表和段表**：索引页号或段号的数据结构，含所有要查找的块的物理地址。
  - **段式系统**：段内位移加上段的物理地址就是最终的物理地址。
  - **页式系统**：只需简单地将页内位移拼接在相应页面的物理地址之后
- **替换算法**： **LRU**
  - 尽可能减少页故障， OS 为每个页面设置“使用位”
- **写策略**：写回法

④ 快表 TLB

- **定义：TLB （ Table Look-aside buffer ）**是一个专用的高速缓冲器，用于存放近期经常使用的页表项 。
  - TLB 中的内容是页表部分内容的一个副本 。
  - TLB 也利用了局部性原理 。
- Alpha Axp 21064 的地址转换过程：Alpha 采用三级页表地址变换过程
- TLB 一般比 Cache 的标识存储器更小、更快

⑤ cache|主存 和 主存|辅存 层次的区别

比较项	“Cache - 主存”次	“主存 - 辅存”次
目的	为了弥补主存**速度**的不足	为了弥补主存**容量**的不足
存储管理实现	主要由**专用硬件**实现	主要由**软件**实现
访问速度比值（一级比二级）	几比一	几百比一
典型块、页大小	几十个字节	几百到几千个字节
CPU对第二级访问方式	可直接访问	均通过第一级
失效时CPU是否切换	不切换	切换到其他进程

---

12. 互连网络基本知识（略考）

① 互联网络定义

- **互联网络**：互连网络是将集中式系统或分布式系统中的结点连接起来所构成的网络，这些结点可能是处理器、存储模块或者其它设备，它们通过互连网络进行信息交换。在拓扑上，互连网络为输入和输出两组结点之间提供一组互连或映象（ mapping ）。

  > 互连网络的拓扑可以采用**静态或动态的结构**。静态网络由点和点直接相连而成，这种连接方式在程序执行过程中不会改变

  - **动态网络**是用开关通道实现的，它可动态地改变结构，使其与用户程序中通信要求匹配。
  - **静态网络**常用来实现一个系统中子系统或计算结点之间的固定连接。动态网络常用于集中式共享存储器多处理系统中。
- **网络规模**：网络可用图来表示。这种图由用有向边或无向边连接的有限个结点构成。其结点数称为网络规模(network size) 。
- **结点度**：与结点相连接的边的数目称为结点度(node degree) 。这里的边表示链路或通道。链路或通道是指网络中连接两个结点并传送数字信号的通路。在单向通道的情况下，进入结点的通道数叫做入度 (in degree) ，而从结点出来的通道数则称为出度 (out degree), 结点度是这两者之和。结点度应尽可能地小并保持恒定。
- **网络直径** : 网络中任意两个结点间最短路径长度的最大值称为网络直径。网络直径应当尽可能地小。
- **等分宽度** : 在将某一网络切成相等两半的各种切法中，沿切口的最小通道边数称为通道等分宽度 b (channel bisection width) 。
- **路由（ routing ）** : 在网络通信中对路径的选择与指定。通常见到的处理单元之间的数据路由功能有移数、循环、置换 (一对一) 、广播(一对全体) 、选播 (多对多) 、个人通信 (一对多) 、混洗、交换等。
- **虫蚀（ wormhole ）** ： 把包进一步分成小片，硬件路由器有片缓冲区，同一个包中所有片象不可分离的同伴一样，以流水方式顺序传送。只有片头包含目标 地址，所有片必须跟随片头。

② 不同网络的特点

- **网格形和环网形**： $N = n^k$ 个结点的 k 维网络的内部结点度为 2k ， 网络直径为 k(n-1) 。纯网络形不是对称的。边结点和角结点的结点度分别为 3 或 2 。 n 为每维上的结点数。
- **二维网格**：两个结点之间的路径数为 K = (X + Y)! / (X! * Y!) ， XY 分别为两结点之间距离增量。对三维网格，两个结点之间的路径数为 K = (X + Y + Z)! / (X! * Y! * Z!) ， XYZ 分别为两结点之间距离增量
- **环形网**：环形网可看做是直径更短的另一种网格。这种拓扑结构将环形和网格组合在一起，并能向高维扩展。环形网沿阵列每行和每列都有环 形连接。一般说来，一个 n×n 二元环网的结点度为 4 ，直径为 2 * [n / 2 ⌋ 。

> 环网是一种对称的拓扑结构，所有附加的回绕连接可使其直径比网格结构减少二分之一

- **超立方体**： 这是一种二元 n- 立方体结构，它已在 n CUBE 和 CM-2 等系统中得到了实现。一般说来，一个 n-立方体由 N = 2n 个结点组成，它们分布在 n 维上，每维有两个结点。
  - 其中参数 n 是立方体的维数， k 是基数或者说是沿每个方向的结点数 (多重性) 。这两个数与网络中结点数 N 的关系为：

$$n = log_k N$$