知识小料

「电计 2203 班」周常规知识整理共享

第 53 期

日期：2025-4-23

学科：计算机系统结构

本文档对《计算机系统结构》（Computer Architecture）课程作出简明复习。

说　明

本文档采用以下方式标识概念/知识点重点等级，星级越高表示概念/知识点越重要。整理基于作者本人看法，请以实际为准。示例如下：

- **记分板算法**：稀有（★★★）概念，要求了解；
- **转移历史表（BHT）**：史诗（★★★★）概念，要求理解；
- **Amdahl 定律**：传说（★★★★★）概念，要求重点掌握与运用。

目录

1 （第 1 章）计算机系统结构概述

1.1 何为「计算机体系/系统结构」

- 引子：按下 PPT 翻页原理
- **计算机层次结构**：最高层为应用程序，其次为操作系统，第三为硬件系统，最底层为硅片工艺。相邻两层之间各有充当「接口」的角色，依次为 API、ISA、工艺模型。
- **冯·诺依曼结构**：数据和程序都在存储器中，CPU 从内存中取指令和数据进行运算，并把结果也放到内存中。

1.2 衡量计算机的指标——性能、价格、功耗

1.2.1 性能

计算性能常用到 CPI 这一定义。

- **CPI**：Cycle Per Instruction，即每条指令所需要的时钟周期数。

专题·CPU 时间计算

- CPU 计算时间公式如下：

$$T_{\text{CPU}} = I_N \times CPI \times T_{\text{c}} \tag{1}$$

  - I_N：指令条数
  - CPI：每条指令的时钟周期数
  - T_{c}：时钟周期

- 若有多个 CPI，平均 CPI 以**使用频次**为权重作加权平均：$\overline{CPI} = \sum_{j=1}^{n} CPI_j \cdot F_j$，其中 $F_j = \dfrac{I_j}{I_N}$（第 j 类指令指令数量占比）。
- 要计算增强后的 CPI，可以使用 $CPI_{\text{新}} = CPI_{\text{原}} - \Delta CPI$ 思路计算。

- **影响 CPU 性能的因素**：

1. 优化算法：影响 I_N
2. 优化编译器：影响 I_N, CPI
3. 优化指令集 (ISA)：影响 I_N, CPI

4. 优化微结构：影响 CPI, T_c
5. 优化主频：影响 T_c

- **计算机性能评价原则**：以基准测试程序为基准，个人机更关注响应时间，工作站更关注吞吐率。
- **基准测试程序**：用以评价计算机性能，常见的有 SPEC 系列程序。

成本、功耗 也是衡量计算机的指标。

1.3 系统结构发展趋势

- **摩尔定律**：晶体管数目每 18–24 个月翻一番；同样晶体管数量的芯片价格下降一半。摩尔定律的阻力的三个阶段的体现：
  1. 晶体管不够用
  2. 存储器速度慢
  3. 晶体管过多而越发难用
- 系统结构发展之「墙」：存储墙、功耗墙；带宽墙、成本墙、应用墙……

1.4 系统结构设计原则

- 系统结构设计的原则：平衡性、局部性、并行性、虚拟化。
- **Amdahl 定律**：计算机部分性能的提高，受制于其时间占比，类似「短板效应」。

专题 · Amdahl 计算题

- 增强比例 F：被改进部分的时间占比，取值 $[0\%, 100\%]$。
- 增强加速比 S：被改进部分改进后与改进前的性能提高倍数，取值大于 1。

$$\text{增强后用时} T_{\text{new}} = T_{\text{old}} \times \left(\frac{1-F}{1} + \frac{F}{S} \right) \tag{2}$$

$$\text{总加速比} S_{\text{overall}} = \frac{T_{\text{old}}}{T_{\text{new}}} = \frac{1}{\frac{1-F}{1} + \frac{F}{S}} \tag{3}$$

理想情况下，$S \to \infty$ 时，最大的总加速比 $S_{\text{overall}} = \dfrac{1}{1-F}$。

1.5 支线 [L01.1]・计算机设计新趋势

- **Top500** 是一个统计高性能计算机性能，并公布前 500 名计算机系统的网站。
- **神威・太湖之光** 是中国国内第一台全部采用国产处理器构建的世界第一的超级计算机，坐落于无锡，于 2022 年在 Top500 网站上斩获第六名。（如今中国最先进的计算机在 Top500 网站上不参评，韬光养晦。）
- 国产 CPU 厂商有：龙芯、鲲鹏、申威、海光等。其中龙芯系列芯片是完全自主知识产权的国产芯片。

2 （第 2 章）指令系统

2.1 指令系统

- **指令系统（ISA）** 是软硬件的交界面，承应用软件与系统软件，启硬件系统与电路器件。它反映了结构设计者对计算机系统的认识。
- 当前主流的两种指令系统为 x86 和 ARM，此外还有 RISC-V、MIPS 等。
- 指令系统的设计原则：兼容性、通用性、高效性、安全性。

2.2 指令系统的演变

- 根据指令长度分类：CISC、RISC、VLIM
- **RISC 指令**：简化指令间关系，有利于指令流水线高效实现。采用 Load-/Store 结构。
- 存储管理的演变：段式、页式、段页式。
- 运行级别的演变：调试模式、主机模式、客户机模式。
- 根据使用数据方式分类：堆栈型、累加器型、寄存器型。

2.3 指令系统的组成

- **大端序**：指数据的高位放在地址的低位。规定网络字节序为大端序。
- **小端序**：指数据的低位放在地址的低位。

地址	2000	2001	2002	2003
大端序	01	23	45	67
小端序	67	45	23	01

表 1：存放 int 型整数 0x01234567

- 寻址方式：支持寄存器寻址、存储器寻址、立即数寻址、比例寻址等。其中比例寻址用于按列访问。
- 指令操作：通常包括算术逻辑指令、访存指令、转移指令、系统指令。
- 指令编码：定长编码（RISC）、变长编码（x86,ARM）、混合编码（IBM,MIPS）。
- RISC 指令系统比较：“亲兄弟”和“表兄弟”。
- C 语言的机器表示。

3 （第 3 章）特权指令系统与 ABI

- **特权指令系统**：指用户和编译器无法触及的那部分指令系统，只用于操作系统等系统软件。
- **应用程序二进制接口（ABI）**：是操作系统对应用程序在这个系统下运行时必须遵守的编程约定。
- **应用程序编程接口（API）**：是一组预先定义的函数或指令集，允许不同的软件应用程序之间进行交互和通信。（易混概念）

4 （第 9 章）指令流水线・上

4.1 支线 [L09.0]・流水线基本概念

- 流水线的时空图
- 流水线的「通过时间」和「排空时间」
- 按反馈回路分：「线性流水线」和「非线性流水线」，前者只有串行连接，后者有反馈回路。

- 按功能分：「单功能流水线」和「多功能流水线」。其中多功能流水线进一步划分：

  1. **静态多功能流水线**：静态流水线的不同的功能（如加法和乘法）不允许在同一时间片内同时出现，必须等一种功能的流水线全部排空才可以运行其他功能。
  2. **动态多功能流水线**：同一时间内，各段可以按照不同的方式连接，同时执行多种功能。允许指令乱序执行。

4.2 流水线处理器

- CPU 数据通路，包括 PC、指令存储器、译码部件、通用寄存器、ALU 等。在此基础上添加模块：

  1. 控制逻辑：位于指令每一级的寄存器处
  2. 有效位标记：位于指令每一级的寄存器处
  3. 寄存器相关判断逻辑
  4. 前递机制

- **流水线冒险**、**指令相关**与**指令冲突**，相关概念如下图 1 所示。此外，相关性的存在只预示着存在有冲突的可能性，不代表一定产生冲突。

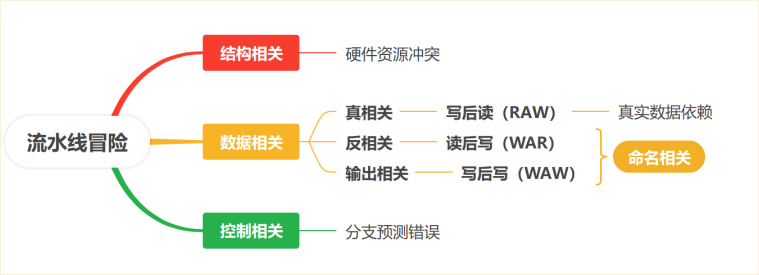

图 1： 流水线冒险概念图

ADD	r_1	r_2	r_3
ADD	r_4	r_1	r_5

(a) 真相关，写后读

ADD	r_4	r_1	r_5
ADD	r_1	r_2	r_3

(b) 反相关，读后写

ADD	r_1	r_2	r_3
ADD	r_1	r_2	r_4

(c) 输出相关，写后写

表 2： 数据相关的例子（注：ADD r_i r_j r_k 表示 $[r_i] \leftarrow [r_j] + [r_k]$）

4.3 静态流水线解决指令相关

明确：在静态流水线中，指令不能乱序执行。

4.3.1 数据相关

- 数据相关包括 RAW、WAW、WAR，其中 WAW 和 WAR 属于命名相关，可以通过寄存器更名的方式解决。RAW 是真实数据依赖相关，需要用阻塞、前递等方法。
- 可以用 **阻塞** （stall）的方式解决数据相关。当相关指令在译码阶段发现与尚未执行完的指令存在数据相关，则阻塞在译码阶段，直到相关指令完成写回阶段。（例如表 2(a) 种情况中，下方指令 ADD r_4 r_1 r_5 需要读寄存器 r_1 的值，那么它在取指阶段完成后会阻塞，直到上方指令 ADD r_1 r_2 r_3 完成写回阶段后，才会继续译码阶段的执行。因此下方指令会在上方指令的执行、访存、写回阶段阻塞，共空 3 拍。参考 PPT《Chapter09_ 指令流水线-1.pptx》第 55 页。）
- 可以用 **前递** （forwarding）的方式解决数据相关。当指令结果产生后（执行阶段或访存阶段完毕），立即将此结果送到后续相关指令的译码阶段中。

4.3.2 控制相关

- 控制相关一般指跳转指令和取指 PC 指令的相关。解决控制相关主要有以下方法：
- 引入流水线阻塞法。如译码阶段遇到分支指令则暂停取指，或取消下一条指令的取回。
- **延迟槽** （delay slot）技术。设立一个延迟槽，存放条件分支指令的**紧接下一条**指令；不管条件分支指令是否跳转，都执行延迟槽中的指令。如果条件的确不满足，再取消延迟槽指令的执行。
- 转移预测。（下文提到）
- 循环展开。（下文提到）

异常处理 （了解）

- 异常多种多样，在指令执行的任一阶段均可能发生。在指令内可恢复异常的处理比较困难。

4.4 支线 [L09.2]・流水线的性能指标

假设 n 条指令，k 段流水，每一段所用时间为 $\Delta t_i (i = 1, 2, \ldots, k)$。

- 流水线 **瓶颈段** ：指流水线耗时最长的一段。
- 流水线 **吞吐率** ：指单位时间内流水线完成的任务数量或输出的结果数量。
- **最大吞吐率** $TP_{\max}$：取决于流水线的瓶颈段所需时间，即

$$TP_{\max} = \frac{1}{\max\limits_i \{\Delta t_i\}} \tag{4}$$

- **实际吞吐率** TP：等于指令条数 n 与总时间 T 之比：

$$TP = \frac{n}{T_{\text{流水}}} = \frac{n}{\sum_{i=1}^{k} \Delta t_i + (n-1) \cdot \max\limits_i \{\Delta t_i\}} \tag{5}$$

- **流水线加速比** S：指同功能下非流水线的用时与流水线的用时之比：

$$S = \frac{T_{\text{非流水}}}{T_{\text{流水}}} = \frac{n \cdot \sum_{i=1}^{k} \Delta t_i}{\sum_{i=1}^{k} \Delta t_i + (n-1) \cdot \max\limits_i \{\Delta t_i\}} \tag{6}$$

- **流水线效率** E（或 η）：指 n 个任务占用的时空区与总时空区之比：

$$\begin{aligned} E &= \frac{\text{有效时空区}}{\text{总时空区}} = \frac{n \cdot \sum_{i=1}^{k} \Delta t_i}{k \cdot T_{\text{流水}}} \\ &= \frac{n \cdot \sum_{i=1}^{k} \Delta t_i}{k \cdot \left(\sum_{i=1}^{k} \Delta t_i + (n-1) \cdot \max\limits_i \{\Delta t_i\} \right)} \end{aligned} \tag{7}$$

专题・流水线时空图作图

时空图有两种：

- 一种是以横轴为时间、纵轴为流水段的时空图，在分析流水线的性能指标时，一般采用这一种。
- 另一种是以横轴为时间、纵轴为指令的时空图，在分析流水线的阻塞情况时，一般采用这一种。

对于前面一种流水线，在作图时应注意：

- 有序作图，最好用直尺；
- 留意是 静态流水线 还是 动态流水线 ；
- 有的流水线只允许部分流水段并行，此时「瓶颈段」可能有变；
- 要留意数据冲突、控制冲突等带来的阻塞。

此外，通常吞吐率 $TP, TP_{\max}$ 含 Δt 项，而加速比 S 和效率 E 不含 Δt 项。

4.5 指令调度技术・循环展开

- 编译器（软件）的静态调度可以有效缓解控制相关。
- 循环展开 技术是静态调度的一种。它通过展开循环扩大基本块、移动不相关指令、寄存器重命名、巧妙设置偏移量等方法，将每次循环从 8 拍降低至 3.5 拍。
- 软件调度和硬件调度的区别：软件调度范围更大，而硬件调度可以掌握一些编译时不明确的相关性信息。

5 （第 9 章）指令流水线・下

5.1 提高流水线效率・动态调度

- 动态调度 解决 **RAW,WAR,WAW 即数据相关**问题，其基本思想是「乱序执行，乱序完成，有序提交」。
- CDC6600 记分板算法 是乱序执行，乱序完成，**乱序提交**的算法，并不能精确中断。

5.1.1 Tomasulo 算法

- **Tomasulo 算法**（原始版）的基本思想仍是「乱序执行，乱序完成，**乱序提交**」，不能精确中断。
- Tomasulo 算法的关键组件有：
  - **保留站**：用来缓存源操作数，跟踪操作数是否就绪。
  - 寄存器重命名：避免 WAR,WAW 阻塞（要求能对 WAW,WAR 寄存器手工重命名）
  - 结果总线 (CDB)：用于广播指令完成的结果。
- Tomasulo 算法的大致流程：指令发射 → 执行 → 写回。

5.1.2 重排序缓存 (ROB)

- **重排序缓存 (ROB)**（reorder buffer）给 Tomasulo 算法打了一个「补丁」，这使得 ROB 加持下的 Tomasulo 算法获得了「乱序执行，乱序完成，**有序提交**」的能力，能够精确中断。
- ROB 在工作时，存储指令执行结果，直到确认无误后更新寄存器，释放 ROB 相应项数据。
- ROB+Tomasulo 算法的大致流程：指令发射 → 执行 → 写回 → 提交。

5.2 提高流水线效率・多发射

5.2.1 支线 [L09.5]・多发射技术

- **多发射技术** (multiple issue)：在每个时钟周期发射不止一条指令。
- 引入多发射技术后，变成「超标量」处理器、「超流水线」，一般采用动态流水线。设计开销更大。
- **IPC**：Instruction Per Cycle，即每一时钟周期发射的指令条数。目标为 CPI < 1 或 IPC > 1，方法有：
  - **超标量**：一次发射多条指令；
  - **超长指令字 (VLIM)**：把几条指令的操作组装成固定格式的长指令；
  - 向量处理机：专用来计算矩阵、张量等多维数据。

  其中**前两种方法**算作多发射技术。

5.3 提高流水线效率・转移预测

- **转移预测**（也称转移猜测）技术解决**控制相关**问题。
- 转移预测在硬件上依靠「转移预测器」实现。转移预测器基于当前 PC 值和过去的转移历史记录，给出转移方向和目标地址。
- 下文提到的都是动态转移预测行为。

5.3.1 转移历史表

- **转移历史表 (BHT)** 用 PC 低位作为索引，记录同一项上次转移是否成功。BHT 有两个属性：多位 PC、1 位转移结果。
- 两位 BHT 表方法：连续两回猜错才会改变猜测方向。其状态图如图 2 所示。
- 两位 BHT 表比一位 BHT 表更好，体现在循环间隙猜测更优。在每个循环间隙中（跳转方向记录中的 101 子段），一位 BHT 表会猜错 2 次，而两位 BHT 表仅猜错 1 次。

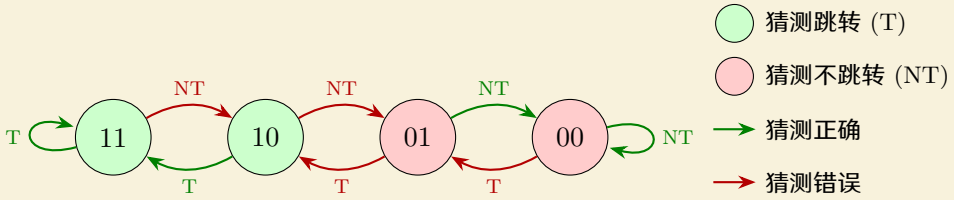

图 2: 两位 BHT 状态图（箭头上标识为实际发生结果）

- **分支目标缓冲 (BTB)**：在跳转前，先对两种结果提前准备，未雨绸缪，失效时再替换。
- **相关分支预测** (correlated branch prediction)：建立一个 (m, n) 预测器，它考察此前的 m 条分支记录（总共 2^m 种可能性），建立 n 位 BHT 表。特别地，$(0, 2)$ 预测器即退化为两位 BHT 表。这样的 (m, n) 预测器总共的位数为：$2^m \times n \times$ 涉及分支条目数。
- 组合分支预测器：类似机器学习中的「集成学习」思想，统合多个预测器的结果进行预测。
- **锦标赛预测法**：两个预测器互搏，任一预测器连续两次错误则换另一个。

5.4 提高流水线效率・高速缓存

- **高速缓存** 技术用于提升 RAW 数据冲突时的访存速度。
- 存储层次基本原理：**程序访问的局部性** 。

5.4.1 主存—Cache 层次

- Cache 映像方式：直接映像、全相连、组相连；
- Cache 失效替换：随机替换、先入先出 (FIFO)、最近最少使用 (LRU)；
- Cache 写命中时的策略：
  1. **写回** (Write Back)：写 Cache 时不写入主存，而当 Cache 数据被替换出去时才写回主存。优点是比较快；而缺点是实现相对复杂。
  2. **写穿透** (Write Through)：写 Cache 时总是同时写入主存。优点是主存中的数据永远是最新的，实现容易；而缺点是比较慢。
- Cache 写失效时的策略：
  1. **写分配** (Write Allocate)：先把失效块读到 Cache，再在 Cache 中写。常搭配「写回」法。
  2. **写不分配** (Write Non-allocate)：直接将失效块写入主存。常搭配「写穿透」法。
- 有时会在「写穿透」的处理器和主存中设置一个 **写缓冲** (Write Buffer)，以平衡写入速度。

5.4.2 Cache 性能分析与优化

- **平均访存时间 (AMAT)** ：指访存阶段的期望时间。若命中则取访问 Cache 时间（以时钟周期为单位）；若不命中则外加惩罚时间乘以命中率，计算期望。AMAT 实际上就是访存指令的 CPI。

$$\mathrm{AMAT} = \mathrm{HitTime} + \mathrm{MissRate} \times \mathrm{MissPenalty} \tag{8}$$

专题・访存时间对计算机性能的影响

如果将 CPU 计算时间划分为非访存指令时间（近似为 ALU 指令）和访存指令时间，那么公式 (1) 中的 CPI 可以稍稍扩展：

$$T_{\mathrm{CPU}} = I_N \times \left(\frac{I_{N\mathrm{ALU}}}{I_N} \times CPI_{\mathrm{ALU}} + \frac{I_{N\mathrm{MEM}}}{I_N} \times AMAT \right) \times T_{\mathrm{c}} \tag{9}$$

其中 $I_{N\mathrm{ALU}}$ 和 $I_{N\mathrm{MEM}}$ 分别为 ALU 指令和访存指令的条数，I_N 为总指令条数，T_{c} 为时钟周期。

$$T_{\mathrm{CPU}} = I_N \times (CPI_{\mathrm{exe}} + \mathrm{MemPerInst} \times \mathrm{MissRate} \times \mathrm{MissPenalty}) \times T_{\mathrm{c}} \tag{10}$$

其中 MemPerInst 为每条指令的平均访存次数。

5.4.3 Cache 失效原因与解决

- **Cache 失效原因 (3C/4C)**：
  1. **冷启动失效**（Cold/Compulsory Miss）：第一次访问时的失效
  2. **容量失效**（Capacity Miss）：因容量有限而被替换的失效
  3. **冲突失效**（Conflict Miss）：不同主存行映射到同一 Cache 块的失效
  4. **一致性失效**（Coherence Miss）：维护「Cache 一致性」导致的失效
- 不同方法降低失效率（Miss Rate）：
  1. 增加块大小（但保持 Cache 容量不变）：减少冷启动失效，但增加容量失效、冲突失效、失效惩罚；
  2. 增加 Cache 容量：减少容量失效，增加访问时间；
  3. 增加相连度、「路猜测」和「伪相连」法：减少冲突失效；
  4. 软件优化（数组合并、循环交换、循环合并、数组分块等）：综合降低失效率
- 降低失效惩罚（Miss Penalty）的方法：优先处理读失效、关键字优先、写合并、牺牲缓存（Victim Cache）、二级 Cache 等。
- 降低命中时间（Hit Time）的方法：简化 Cache 设计、增加 Cache 访问流水级、提高 Cache 访问并行性。

5.5 支线 [L09.6]・I/O 处理与存储系统

- **南北桥**：
  1. 北桥快，连接 CPU，连接内存，连接 GPU；
  2. 南桥慢，连接 I/O。
- 反映存储外设可靠性能的参数：
  1. 系统可靠性（Reliability），用 MTTF 衡量；
  2. 系统可用性（Availability），用 $\frac{\text{MTTF}}{\text{MTBF}}$ 衡量；
  3. 系统可信性（Dependability），不可度量。
  4. **平均无故障时间 (MTTF)**；
  5. **平均故障间隔时间 (MTBF)** $= \text{MTTF} + \text{MTTR}$；
  6. **平均修复时间 (MTTR)**。
- 「廉价磁盘冗余阵列 (RAID)」提高了系统可用性。

6 （第 10 章）并行编程基础

6.1 三种并行行为

- **指令级并行 (ILP)**：不相关的指令可以并行执行
- **数据级并行 (DLP)**：对多个元素同时执行相同操作
- **任务/线程级并行 (TLP)**：将不同任务（进程/线程）分布到不同处理单元并行执行

开发指令级并行的两种方法是：

1. **基于硬件的动态开发方法**：也就是《指令流水线・下》介绍的四种方法：动态调度、多发射、转移预测、高速缓存。
2. **基于软件的静态开发方法**：依赖于编译器在编译时对程序的分析与优化，例如循环展开。

6.2 Flynn 分类法

- **Flynn 分类法**：根据**指令流**和**数据流**的多倍性对计算机系统结构进行分类。目前分成三种：

1. **单指令流单数据流（SISD）**
2. **单指令流多数据流（SIMD）**
3. **多指令流多数据流（MIMD）**

- 对于 MIMD 系统，还能按照信息传递方法分成：

  1. 多处理机系统：基于共享存储器（Shared Memory）
  2. 多计算机系统：基于消息传递（Message Passing）

  其中内存组织方式还能细分为

  1. 中心化内存（Global/Centralized Memory）
  2. 分布式内存（Distributed Memory）

- 因此还有一种 **Johnson 分类法**，对 MIMD 系统进行分类：

  1. 中心化内存 + 共享存储器（GMSV）
  2. 分布式内存 + 共享存储器（DMSV）
  3. 分布式内存 + 消息传递（DMMP）

6.3 典型并行计算机结构与并行编程

- 典型并行计算机结构：

  1. 对称多处理机（SMP）
  2. 工作站机群（COW）
  3. 大规模并行处理机（MPP）

- 典型并行编程环境：

  1. SIMD 数据并行编程
  2. **POSIX Threads 标准**，简称 Pthread，IEEE 官方标准
  3. **OpenMP** 共享存储编程标准
  4. **MPI** 标准

- 「并行编程的挑战」——联系第一章 Amdahl 定律计算和 CPI 计算

7 多核处理器

多核处理器可以这么分类：异构、同构；通用、专用；重核、轻核；多核、众核……要求通用多核处理器访存带宽和峰值性能差距不能太大。

7.1 分支 [L11.1]・单处理器时代的终结

- 单核处理器性能提升的瓶颈：频率墙、功耗墙、存储墙、应用墙。传统性能提升手段失效（如流水线、超标量等），主频增长停滞。
- 多核处理器成为必然趋势。
- 「新摩尔定律」：性能提升由「更快」转为「更宽」。多核 (TLP) 与数据级并行 (DLP) 将成为未来主流。
- FPGA 优势、RAMP 并行计算研究项目。

7.2 分支 [L11.2]・同时多线程

- **多线程** ：在一个处理器上运行多个线程。
- 多线程的分类，如图 3 所示：
  1. **细粒度** ：线程切换于每条指令之间。采用「轮叫调度」。
  2. **粗粒度** ：线程切换于代价高、时间长的阻塞出现之时。
- **同时多线程 (SMT)** ：是一种在多流出、动态调度处理器上同时开发线程级并行 (TLP) 和指令级并行 (ILP) 的改进的多线程技术。
- 换言之，同时多线程允许**单个**物理 CPU 核心同时执行**多个**线程的指令。通过共享执行资源（如 ALU、寄存器等）实现。它只有在细粒度实现方式下才有意义。

7.3 访存结构、存储一致性模型

- 通用多核 CPU 的**片上 Cache 结构** ：片内共享最后一级 Cache（Last Level Cache,LLC），其他级别的 Cache 由片内私有，片间共享内存。
- 共享 LLC 结构细分为：集中式共享 (UCA)、分布式共享 (NUCA)。
- **存储一致性模型** ：
  1. **顺序一致性** ：并行执行结果应等于多进程环境下的一个执行结果。[1]

[1] 这类似于《数据库系统原理》中的「可串行化调度」：多个事务的并发执行是正确的，当且仅当其结果与按某一次序串行地执行这些事务时的结果相同。

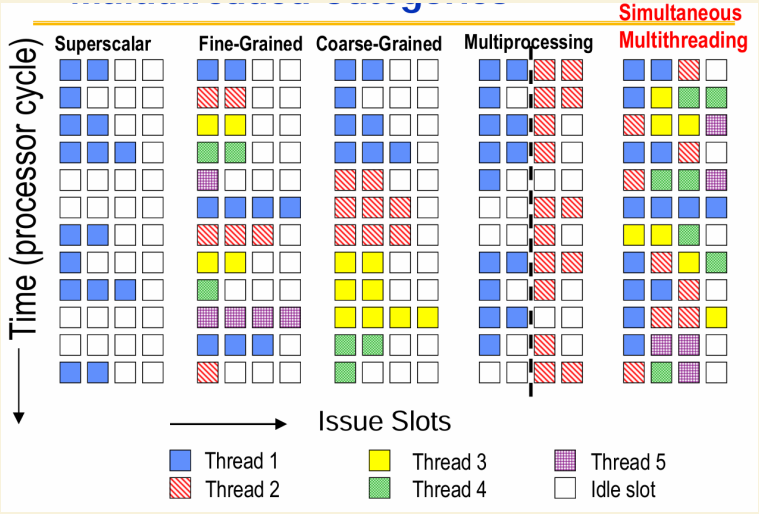

图 3: 多线程分类（从左到右依次为超标量、细粒度、粗粒度、多处理器、同时多线程）

2. **处理器一致性**：在任一取数操作 LOAD 允许被执行之前，所有在同一处理器中先于这一 LOAD 的取数操作都已完成；在任一存数操作 STORE 允许被执行之前，所有在同一处理器中先于这一 STORE 的访存操作（包括 LOAD 和 STORE）都已完成。
3. **弱一致性**：只在需要同步的同步点维护一致性。
4. **释放一致性**：把同步操作细分为获取和释放操作。

- **Cache 一致性协议**：在多处理器系统中，同一个内存地址在不同 Cache 中的数据副本保持一致。有两种策略用于维护 Cache 一致性：

  1. **写使无效**（Write Invalidate）：当某处理器对共享数据进行写操作时，将其他 Cache 中的该数据副本标记为无效，后续其他处理器需重新读取最新值。
  2. **写更新**（Write Update）：当某处理器修改共享数据时，立即将更新后的值广播到所有持有该数据副本的 Cache，保持多副本一致。

- 写使无效和写更新是为了解决「如何传播新值」的问题。「侦听协议」和「目录协议」是为了解决「新值传播给谁」的问题。

7.4 互连结构、同步机制

- **片上互联网络 (NOC)** 是一种典型的互联结构。它具有片上总线、交叉开关、拓扑结构、路由算法、路由表、流量控制等计算机网络都有的机制。
- **同步机制** ：三种常见同步机制：

  1. 互斥锁操作
  2. 路障操作
  3. 事务内存

  对于上述机制，有三种常见实现方式：读—改—写指令、LL/SC 指令、用户级软件例程。

名词索引