计算机体系结构
Computer Architecture

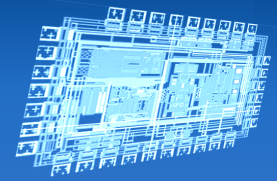
第一讲：绪论

南京大学　　计算机科学与技术系

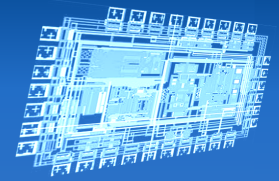
2023年春季学期

Part of these slides are adapted from CMU 18-447 slides of Prof. Onur Mutlu

什么是计算机体系结构?



计算机体系结构的定义



Computer Architecture

Computer architecture, like other architecture, is the art of determining the needs of the user of a structure and then designing to meet those needs as effectively as possible within economic and technological constraints.

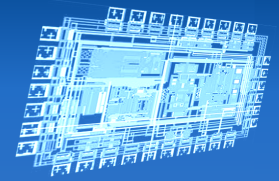
– Frederick P. Brooks Jr, Planning a Computer System: Project Stretch, 1962 [1]

传统定义

The term architecture is used here to describe the attributes of a system as seen by the programmer, i.e., the conceptual structure and functional behavior as distinct from the organization of the dataflow and controls, the logic design, and the physical implementation.

– Gene Amdahl, IBM Journal of R&D, April 1964 [2]

计算机体系结构的定义



Computer Architecture

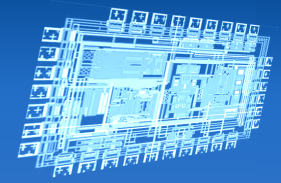
Computer architecture, like other architecture, is the art of determining the needs of the user of a structure and then designing to meet those needs as effectively as possible within economic and technological constraints.

– Frederick P. Brooks Jr, Planning a Computer System: Project Stretch, 1962 [1]

传统定义

The term architecture is used here to describe the attributes of a system as seen by the programmer, i.e., the conceptual structure and functional behavior as distinct from the organization of the dataflow and controls, the logic design, and the physical implementation.

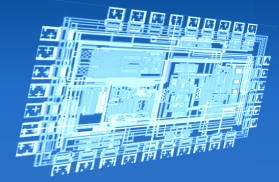
– Gene Amdahl, IBM Journal of R&D, April 1964 [2]



建筑师与架构师（Architect)

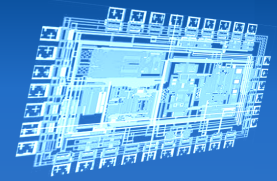
- 流水别墅（Falling Water）
  - 1937
  - Pennsylvania
- 建筑师（Architect）
  - Frank Lloyd Wright
- 艺术与工程的结合

建筑师与架构师（Architect)



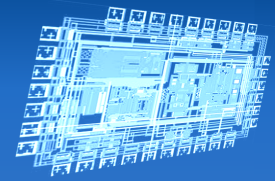
- 流水别墅（Falling Water）
  - 1937
  - Pennsylvania
- 建筑师（Architect）
  - Frank Lloyd Wright
- 艺术与工程的结合

架构师/建筑师的自我修养



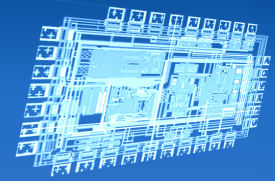
- 全面的素质要求
  - 长期的持续努力、专注
  - 长期经验：无论成功或者失败的经验
  - 跳出既定框架的创造性思维
  - 掌握设计的基本原则
  - 理解历史上著名设计的成功或失败的原因
  - 判断力和直觉
  - 熟练的技巧 (数学,结构, 艺术, ...)
  - ...

架构师/建筑师的自我修养



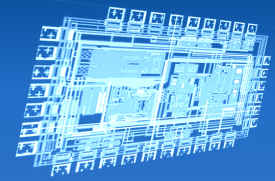
- 全面的素质要求
  - 长期的持续努力、专注
  - 长期经验：无论成功或者失败的经验
  - 跳出既定框架的创造性思维
  - 掌握设计的基本原则
  - 理解历史上著名设计的成功或失败的原因
  - 判断力和直觉
  - 熟练的技巧 (数学,结构, 艺术, ...)
  - ...

架构师/建筑师的自我修养



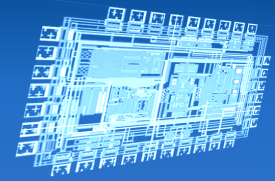
- 全面的素质要求
  - 长期的持续努力、专注
  - 长期经验：无论成功或者失败的经验
  - 跳出既定框架的创造性思维
  - 掌握设计的基本原则
  - 理解历史上著名设计的成功或失败的原因
  - 判断力和直觉
  - 熟练的技巧 (数学,结构, 艺术, ...)
  - ...

架构师/建筑师的自我修养



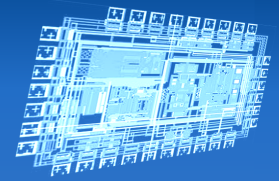
- 全面的素质要求
  - 长期的持续努力、专注
  - 长期经验：无论成功或者失败的经验
  - 跳出既定框架的创造性思维
  - 掌握设计的基本原则
  - 理解历史上著名设计的成功或失败的原因
  - 判断力和直觉
  - 熟练的技巧 (数学,结构, 艺术, ...)
  - ...

架构师/建筑师的自我修养



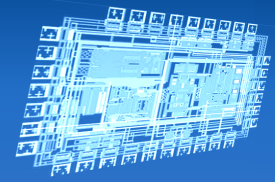
- 全面的素质要求
  - 长期的持续努力、专注
  - 长期经验：无论成功或者失败的经验
  - 跳出既定框架的创造性思维
  - 掌握设计的基本原则
  - 理解历史上著名设计的成功或失败的原因
  - 判断力和直觉
  - 熟练的技巧 (数学,结构, 艺术, ...)
  - ...

架构师/建筑师的自我修养



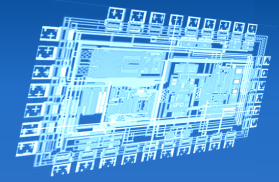
- 全面的素质要求
  - 长期的持续努力、专注
  - 长期经验：无论成功或者失败的经验
  - 跳出既定框架的创造性思维
  - 掌握设计的基本原则
  - 理解历史上著名设计的成功或失败的原因
  - 判断力和直觉
  - 熟练的技巧 (数学,结构, 艺术, . . .)
  - . . .

架构师/建筑师的自我修养

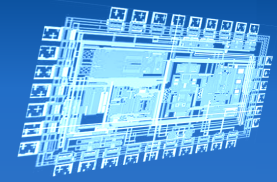


- 全面的素质要求
  - 长期的持续努力、专注
  - 长期经验：无论成功或者失败的经验
  - 跳出既定框架的创造性思维
  - 掌握设计的基本原则
  - 理解历史上著名设计的成功或失败的原因
  - 判断力和直觉
  - 熟练的技巧 (数学,结构, 艺术, . . .)
  - . . .

架构师/建筑师的自我修养

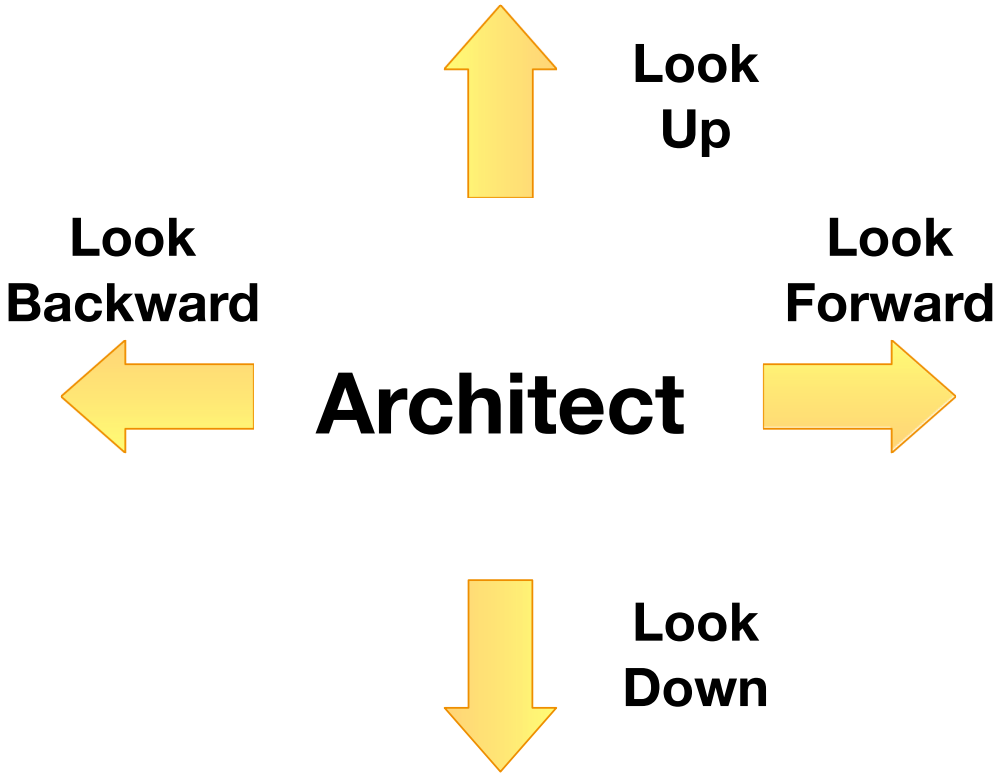


- 全面的素质要求
  - 长期的持续努力、专注
  - 长期经验：无论成功或者失败的经验
  - 跳出既定框架的创造性思维
  - 掌握设计的基本原则
  - 理解历史上著名设计的成功或失败的原因
  - 判断力和直觉
  - 熟练的技巧 (数学,结构, 艺术, ...)
  - ...



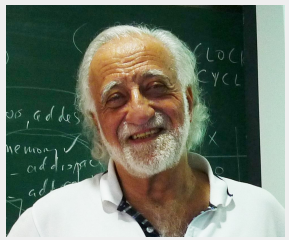
计算机架构师

计算机架构师的职责

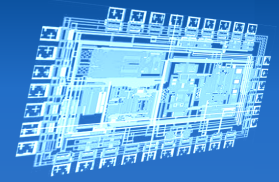


Yale N. Patt

- Professor, UT Austin
- IEEE Fellow, ACM Fellow

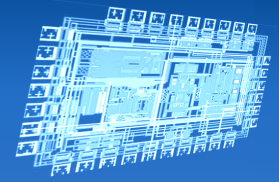


计算机架构师的职责



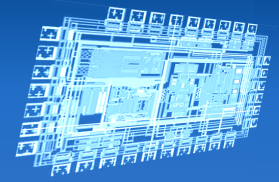
- Look backward (面向过去)
  - 分析前人的设计和设计历史，当年为何这样设计、有哪些坑
- Look forward (面向未来)
  - 规划未来的架构，分析未来潜在的技术革新及需求
  - 在现有技术条件下寻求设计突破
- Look up (面向上层应用)
  - 理解上层应用的需求及关键问题
  - 面向应用的真正问题来进行架构设计
- Look down (面向底层技术)
  - 理解底层物理器件技术及能力限制

计算机架构师的职责



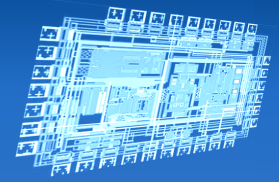
- Look backward (面向过去)
  - 分析前人的设计和设计历史，当年为何这样设计、有哪些坑
- Look forward (面向未来)
  - 规划未来的架构，分析未来潜在的技术革新及需求
  - 在现有技术条件下寻求设计突破
- Look up (面向上层应用)
  - 理解上层应用的需求及关键问题
  - 面向应用的真正问题来进行架构设计
- Look down (面向底层技术)
  - 理解底层物理器件技术及能力限制

计算机架构师的职责



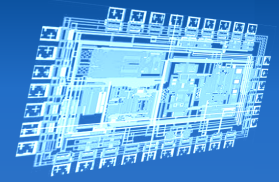
- Look backward (面向过去)
  - 分析前人的设计和设计历史，当年为何这样设计、有哪些坑
- Look forward (面向未来)
  - 规划未来的架构，分析未来潜在的技术革新及需求
  - 在现有技术条件下寻求设计突破
- Look up (面向上层应用)
  - 理解上层应用的需求及关键问题
  - 面向应用的真正问题来进行架构设计
- Look down (面向底层技术)
  - 理解底层物理器件技术及能力限制

计算机架构师的职责



- Look backward (面向过去)
  - 分析前人的设计和设计历史，当年为何这样设计、有哪些坑
- Look forward (面向未来)
  - 规划未来的架构，分析未来潜在的技术革新及需求
  - 在现有技术条件下寻求设计突破
- Look up (面向上层应用)
  - 理解上层应用的需求及关键问题
  - 面向应用的真正问题来进行架构设计
- Look down (面向底层技术)
  - 理解底层物理器件技术及能力限制

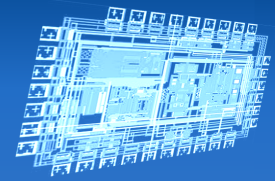
计算机架构师的职责



成为架构师是一条相对艰难的道路，但是值得尝试：

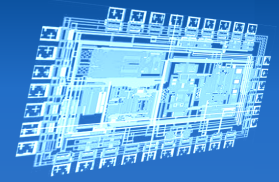
- 国家战略需求
  - 芯片设计
  - 底层系统软件，操作系统，编译器等
  - 工业基础软件
- 个人发展需求
  - 更加广阔的发展前景
  - 具有挑战性的工作内容
  - 造计算机，而不是用计算机，跳出“内卷”！

课程目标



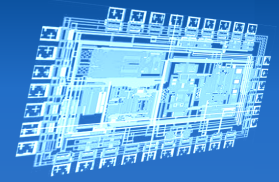
- 理解计算机体系结构的设计基本原则
- 理解计算机架构演进及历史设计方案的优缺点
- 进一步加深对计算机软硬件的整体理解，在理解的基础上实现软硬件协同设计
- 为未来通过实践及科研的持续学习打下基础

课程目标



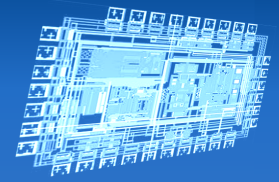
- 理解计算机体系结构的设计基本原则
- 理解计算机架构演进及历史设计方案的优缺点
- 进一步加深对计算机软硬件的整体理解，在理解的基础上实现软硬件协同设计
- 为未来通过实践及科研的持续学习打下基础

课程目标



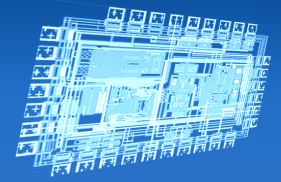
- 理解计算机体系结构的设计基本原则
- 理解计算机架构演进及历史设计方案的优缺点
- 进一步加深对计算机软硬件的整体理解，在理解的基础上实现软硬件协同设计
- 为未来通过实践及科研的持续学习打下基础

课程目标



- 理解计算机体系结构的设计基本原则
- 理解计算机架构演进及历史设计方案的优缺点
- 进一步加深对计算机软硬件的整体理解，在理解的基础上实现软硬件协同设计
- 为未来通过实践及科研的持续学习打下基础

课程基本信息

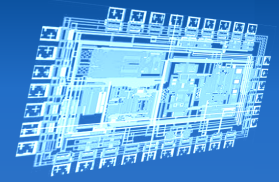


- 上课时间地点
  - 周五下午 14:00 – 17:00
    仙II-110
- 成绩构成
  - 课堂平时成绩: 10%
  - 作业: 20%
  - 实验: 20%
  - 课内讲解及文献调查: 10%
  - 期末考试: 40%
- 课程网站 （教学立方）
  课程PPT、作业、文献资料及视频
  - https://teaching.applysquare.com/
  - 邀请码: P3CTBDQ6

计算机体系结构2...
群号：299754352

QQ: 299754352

课程安排

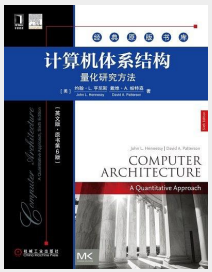


教科书- H&P [3]

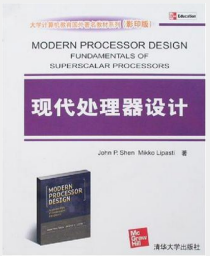
Computer Architecture – A Quantitative Approach

John L. Hennesssy and David A. Patterson

计算机体系结构:量化研究方法 (第六版)
英文版 机械工业出版社



参考资料: 现代处理器设计 影印版 清华大学出版社
S&L: John P. Shen & Mikko Lipasti [4]

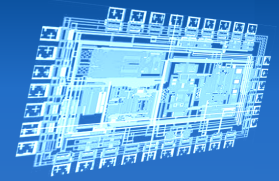


其他资料: P&H [5], P&P [6]

教学团队

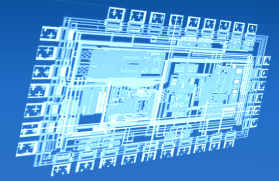
教师	王炜	助教	程文
Email	ww@nju.edu.cn	QQ	625026538
办公室	计算机系楼606		
办公时间	周四下午 14:00 – 16:00		

课程内容



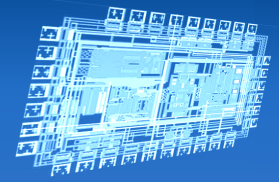
- 计算机体系结构基础
- 指令集设计（Instruction Set Principles）
- 高级流水线技术
- 动态指令调度（Dynamic Scheduling）
- 静态指令调度（Static Scheduling）以及VLIW
- 数据级并行SIMD and GPU
- Cache及主存（Memory）
- 多处理器
- 网络及数据中心

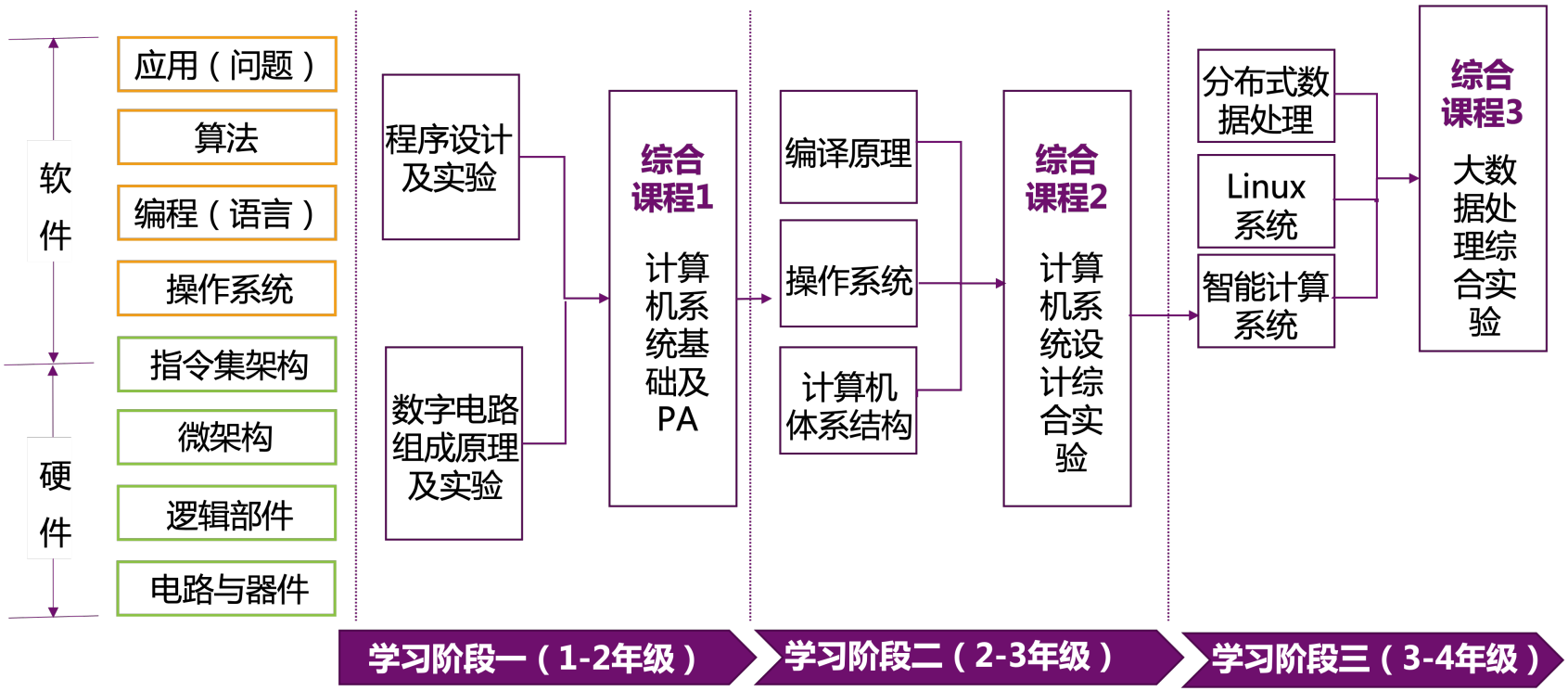
平时作业及实验



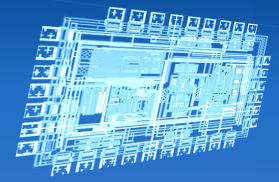
- 实验
  - gem5仿真器
  - 基于ISPC的并行代码优化
- 作业
  - 大约3–4次书面作业
- 课后文献调查
  - 两类文献调查（视选课人数进行后续安排）
  - A: 现代处理器微架构论文
  - B：近年计算机体系结构相关论文
    1. 深度学习硬件加速
    2. RDMA and GPUDirect
    3. 异构技术
    4. 分布式系统
    5. 移动设备及边缘计算
  - 考核方式
    - 提交报告
    - 课内讲解 (∼ 20分钟)

本课程在课程体系中的位置



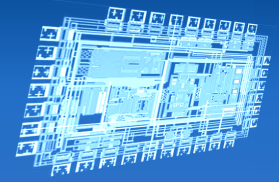


先导课程



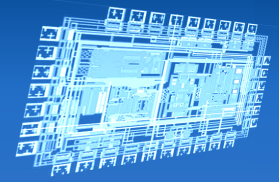
- 程序设计基础
- 计算机组成原理或计算机系统基础
  - RISC-V或MIPS指令集及CPU基本结构
  - 流水线结构
  - Cache
  - 主存
- 操作系统（可选）
  - 虚拟内存（Virtual Memory）
  - 并发及互斥
- 编译原理（可选）
  - 机器无关代码优化（Machine-Independent Optimization）

绪论：计算机体系结构简介



- 阅读资料
  - [3] H&P, 第一章
  - [7] Yale Patt, “Requirements, Bottlenecks, and Good Fortune: Agents for Microprocessor Evolution,” Proceedings of the IEEE 2001
  - [8] John L. Hennessy and David A. Patterson, “A New Golden Age for Computer Architecture,” Communications of the ACM, Feb, 2019
- 本讲内容
  - 计算机系统层次结构
  - 性能指标（Performance Metrics）
  - 设计中的权衡（Tradeoffs)
  - 量化设计方法（Quantitative Approach）
  - 体系结构总体发展趋势

计算机系统层次结构

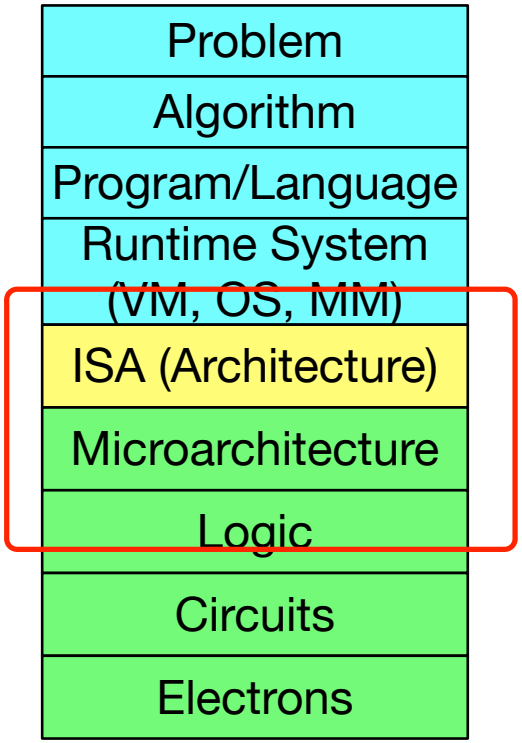


- 分层设计（Layered Design）

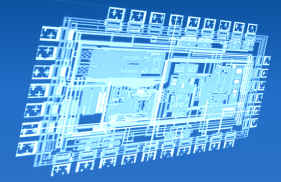
  从实际应用到电路设计分为多个层次

  本课程集中在指令集架构（ISA)及微架构（Microsarchitecture)

- 为什么要分层设计?
  - 设计原则 – 分而治之（Divide and conquer）
  - 设计原则 – 清晰而明确的接口定义
    - 同一接口允许不同实现方式
    - 便于排查问题，确定责任

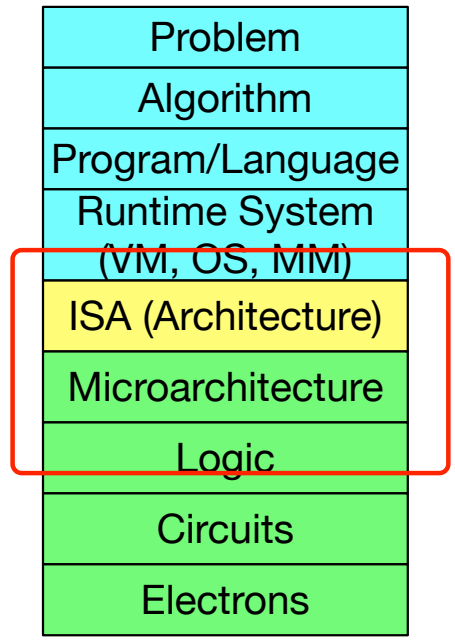


架构与微架构 ISA vs. Microarchitecture

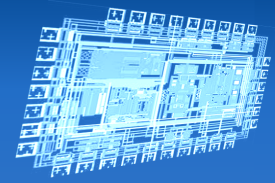


- 指令集架构
  Instruction Set Architecture (ISA)
  软件和硬件之间的接口，软件或者程序员可见的
  - 软件通过指令来指示处理器需要执行的操作
  - 处理器严格按照指令操作说明来进行状态转换
  - 例如：`lw x1, 12(x5)`
    - 具体执行哪些操作
    - 会改变哪些寄存器及Memory?
    - 异常时应该如何处理?

- 微架构
  Microarchitecture (uarch)
  - ISA的具体实现硬件架构

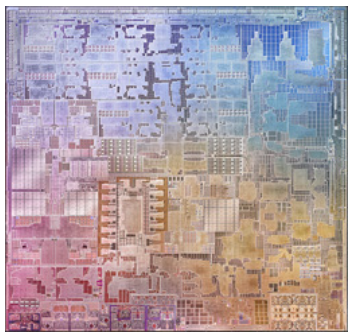


同样的架构可以用不同微架构实现



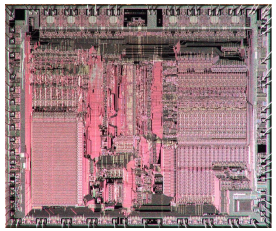
Apple M1

- ARM Instruction Set, 64 Bit
- SoC，含8核GPU及AI核
- CPU 4 Firestorm/4 Icestorm
- 192KB L1 (I) 128KB L1 (D)
- 12MB L2 Cache (F) 4M (I)
- 3.2GHz/14W(F) 2GHz/1.3W(I)
- Out-of-order, 8-wide decode, 630 ROB
- 5nm 16B transitors
- 升级版M2，提升主频，晶体管数，+18%CPU



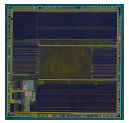
Broadcom BCM2835

- ARM, 32 bit
- SoC含GPU和RAM
- Single core
- 16KB L1 Cache
- 128KB L2 Cache
- ∼1.5W
- 8-stage pipleine
- 700MHz

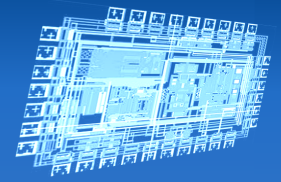


ST STM32F103

- ARM, 32 bit
- Microcontroller with 96KB RAM
- Single core
- ∼30mW
- In-order
- 72MHz

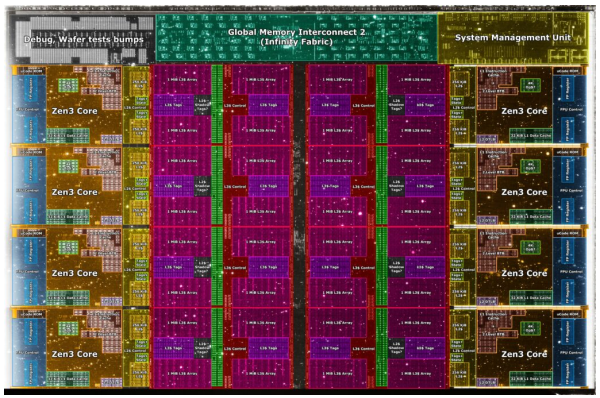


不同指令集，不同微架构，类似的设计



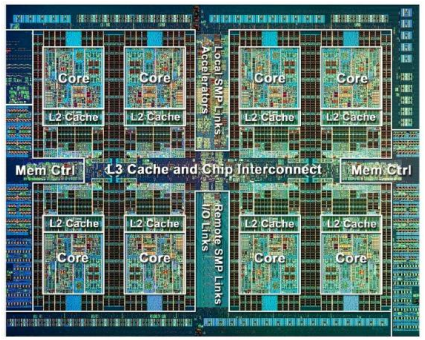
AMD Ryzen 5900X

- X86 Instruction Set
- 16 Core, 32 Thread
- 105W
- 32KB L1 I Cache, 32B L1 D Cache
- 512KB L2 Cache
- Out-of-order, 10 issues/cycle, 256 ROB
- 3.4GHz

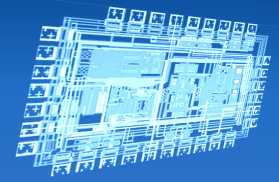


IBM POWER7

- Power Instruction Set
- Eight Core
- 200W
- Decode 6 Instructions/Cycle/Core
- 32KB L1 I Cache, 32KB L1 D Cache
- 256KB L2 Cache
- Out-of-order
- 4.25GHz



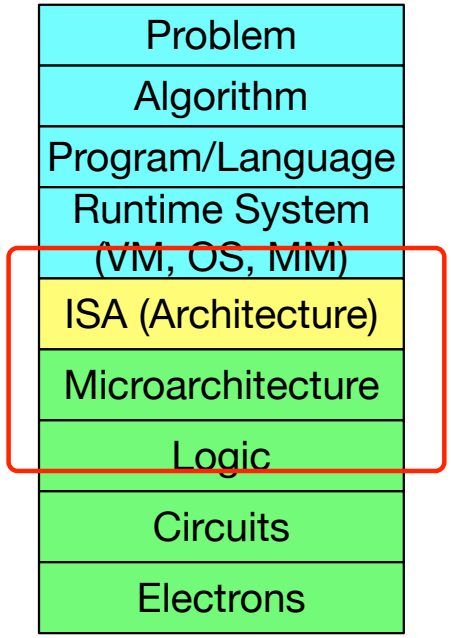
ISA vs. Microarchitecture



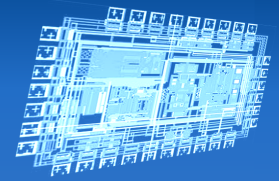
- 哪些是在指令集中规定的，哪些是在微架构里？

  关键区分：程序员/软件/编译器是否可见

  - 指令编码方式
  - 内存页面大小
  - 缓存（Cache）的级数和各级缓存的大小
  - CPU流水线级数
  - 指令执行所需的CPU时钟周期数
  - 中断的类型
  - 是否有缓存预取（prefetch）功能
  - 是否有分支预测（Branch prediction）功能
  - ...



跨层设计（Cross Layer Design）

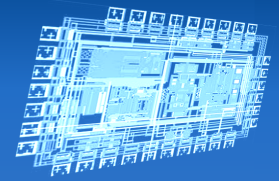


分层设计原则使系统设计/实现者每次只关注于一个特定问题。

但是，架构师应该能够自如地跨越层次限制：
为什么？

- 系统出现各类问题时，有可能是软件问题，有可能是硬件问题，需要跨层解决
- 同样的需求，可以用软件来实现也可以用硬件实现
  - 跨层优化的性能总是不差于分层优化：将一个大问题分解为多个小问题，每个小问题进行最优化，其解空间总是被包含在大问题整体优化的解空间内的

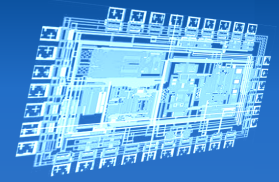
性能指标（Performance Metrics）



系统设计面向多种性能指标，新手往往忽视某些指标或者混淆指标的含义

- 性能（Performance）
  - 处理速度（Execution time）
  - 吞吐量（Throughput）
- 功耗（Power）与能耗（Energy）
- 成本（Cost）
  - CAPEX: CAPital EXpenditure
  - OPEX: OPerating EXpense
- 可靠性（Dependability）
  - MTTF: Mean Time To Failure
  - MTBF: Mean Time Between Failure

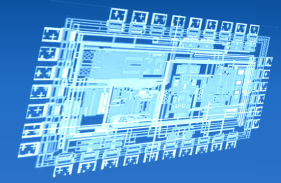
设计中的权衡（Tradeoffs in Designs）



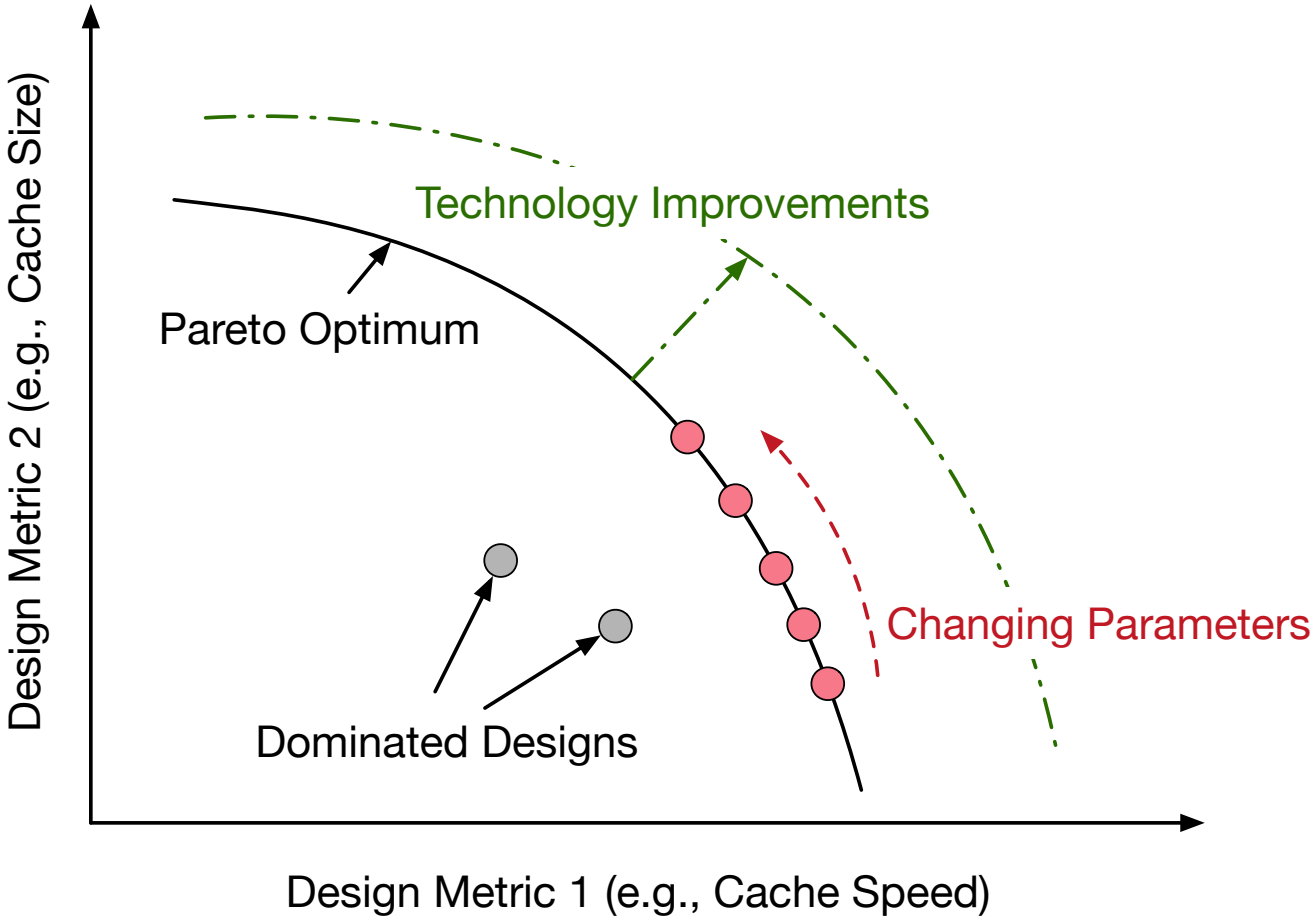
世界上没有一个“完美”的设计

- 特定的设计选择总是有优点和缺点的
- 提升某一方面的指标往往伴随着其他方面指标的削弱

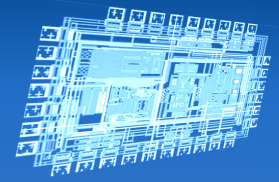
  例如：缓存大小增加后，缓存电路规模变大，需要更长的连线并驱动更多单元，必然带来缓存访问时间的延长
- 架构师的重要职责之一：权衡



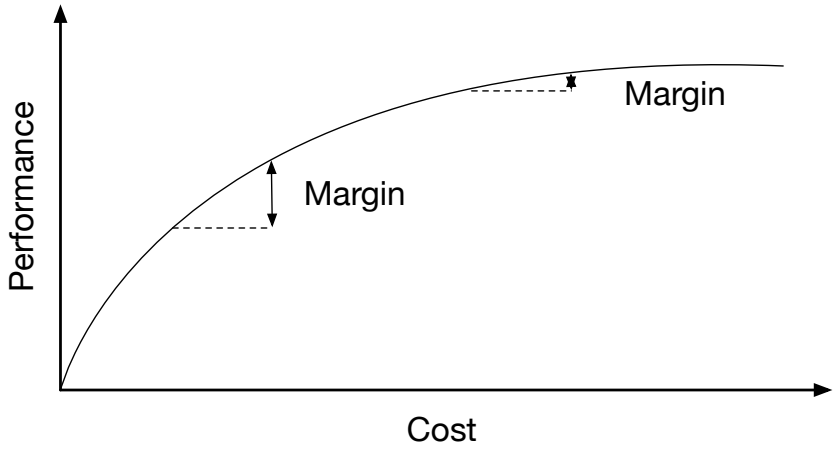
理解设计边界


Technology Improvements
Pareto Optimum
Changing Parameters
Dominated Designs
Design Metric 2 (e.g., Cache Size)
Design Metric 1 (e.g., Cache Speed)

权衡的基本原则

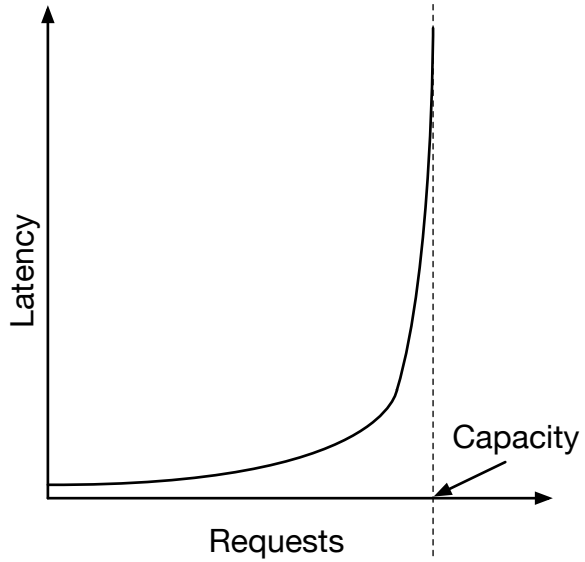


- 边际效应递减（Diminishing Marginal Gain）

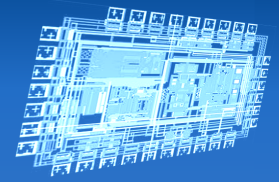


- 系统临界点

  Chock Points under congestion



权衡示例



如何选择一个均衡点

- 假设需要优化N个不同的指标x_i
- 目标：最大化乘积（不能有短板）

$$x_1 \times x_2 \times \ldots \times x_N$$

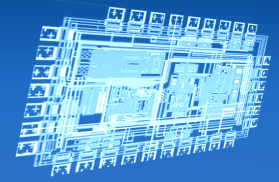
- 等价于最大化: $\sum_i \log x_i$
- 成本限制: $\sum_i c_i x_i = B$

c_i – 提升第i项指标的单位代价

B – 总的成本限制（Budget）

- 等价于比例公平问题（proportional fairness），最优解: $c_i x_i = \frac{B}{N}$

权衡示例



如何选择一个均衡点

- 假设需要优化N个不同的指标x_i
- 目标：最大化乘积（不能有短板）

$$x_1 \times x_2 \times \ldots \times x_N$$

- 等价于最大化: $\sum_i \log x_i$
- 成本限制: $\sum_i c_i x_i = B$

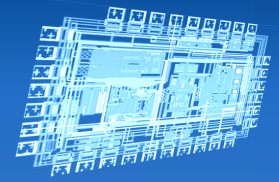
c_i – 提升第i项指标的单位代价

B – 总的成本限制（Budget）

- 等价于比例公平问题（proportional fairness），

最优解: $c_i x_i = \frac{B}{N}$

量化设计（Quantitative Approach）



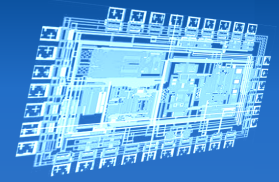
- 如何对比和分析不同的设计?

  定性 Qualitative vs. 定量 Quantitative

  艺术 Art vs. 科学 Science

- 基本的量化分析方法
  - 快速估算Back of the envelope calculations
  - 仿真

量化分析



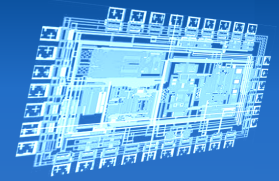
- 夫未战而庙算胜者，得算多也，未战而庙算不胜者，得算少也。多算胜，少算不胜，而况于无算乎！

  —孙子兵法 计篇

- 选将、量敌、度地、料卒、远近、险易。计于庙堂也。

  —曹操

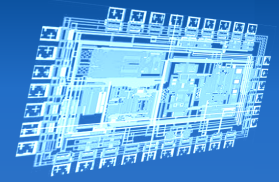
快速估算



- 面试常见问题
- 估算技巧 – *Programming Pearls [9]*
  - 简单而有效的模型
  - 不同方式计算，相互验证
  - 保留安全余量
- 基本原则

Everything should be made as simple as possible, but no simpler.

快速估算

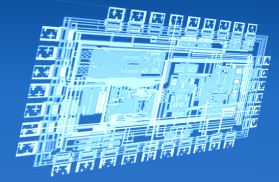


- 面试常见问题
- 估算技巧 – *Programming Pearls [9]*
  - 简单而有效的模型
  - 不同方式计算，相互验证
  - 保留安全余量

- 基本原则

Everything should be made as simple as possible, but no simpler.

快速估算的例子



CPI

CPI: Clock cycles Per Instruction.

$$CPI = \frac{CPU\ clock\ cycles\ for\ a\ program}{Instruction\ count}$$

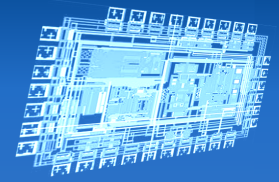
分析两个处理器哪一个在执行以下给定指令比例的程序更快?

参数	处理器A	处理器B
主频	1GHz	2GHz
ALU指令CPI	1	2
分支指令CPI	2	3
访存指令CPI	1	3

指令比例:

- 50% 算术（ALU）指令
- 10% 分支（Branch）指令
- 40% 访存（Memory）指令

Example



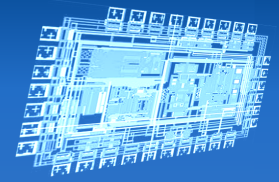
分析两个处理器哪一个在执行以下给定指令比例的程序更快?

参数	处理器A	处理器B
主频	1GHz	2GHz
ALU指令CPI	1	2
分支指令CPI	2	3
访存指令CPI	1	3

指令比例:
50% ALU指令, 10% 分支指令, 40% 访存指令

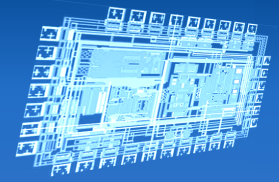
解:
处理器A: 平均CPI $= 0.5 \times 1 + 0.1 \times 2 + 0.4 \times 1 = 1.1$
处理器B: 平均CPI $= 0.5 \times 2 + 0.1 \times 3 + 0.4 \times 3 = 2.5$
处理器A: 每秒$1/1.1 = 0.909$G条指令, 处理器B: 每秒$2/2.5 = 0.8$G指令。处理器A较好。

仿真评估



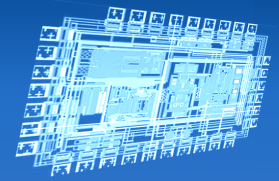
- 利用仿真技术，按照给定模型实现模拟程序来快速判断给定设计针对特定应用的性能
- 主要挑战：
  - 针对特定负载（某些特定程序或者benchmark），无法用简单模型进行计算
  - 系统性能涉及到内部实现，例如，Cache大小、分支预测算法等等细节
- 设计目标
  - 仿真速度（Speed）：跑完特定仿真所需时间
  - 灵活性（Flexibility）：是否易于修改和添加新的算法
  - 准确性（Accuracy）：仿真结果是否和真实硬件能够保持一致

仿真器设计中的权衡



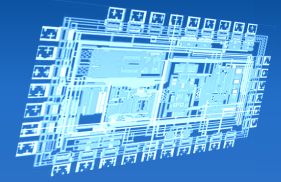
- 速度、灵活性和准确性三个指标是存在一定相互矛盾的，需要进行取舍。
- 例如，采用高层次仿真器，抽象化电路细节，只考虑简单的指令执行逻辑及缓存大小模型
  - 优点
    - \+ 可以运行实际的软件负载
    - \+ 运行速度较快，能够快速获取性能的大致趋势
  - 缺点
    - \- 有可能不够准确，对硬件行为的描述与实际行为有偏差
    - \- 某些特定硬件实现细节带来的性能变化可能无法模拟

计算机体系结构发展趋势



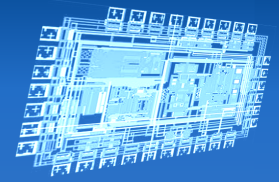
- 摩尔定律（Moore's Law [10]）维持了近50年
  - 半导体技术的进步带来了持续的性能及功耗的提升，软件从业者可以在无需改变编程模型的条件下享受性能红利
- 半导体技术在过去十到二十年发展明显放缓
  - Dennard scaling基本停止，电压无法继续降低
  - 摩尔定律放缓或者已经停止
  - 在可见的未来还没有可以替代CMOS的工艺
  - 功耗已经成为重要瓶颈
- 程序员再也无法直接享受"免费午餐"，而是要考虑
  - 并行编程
  - 异构系统编程

目标用户发生变化



- 移动设备（手机/可穿戴设备/传感器）
  - ARM销量远超过X86处理器
  - 典型芯片采用system-on-a-chip (SoC)方式集成通用ARM核，并扩展自定义硬件外围设备
  - 集成各类通信设备、传感器控制、加速器及GPU
  - 功耗和集成度是设计关键
- 数据中心服务器（Warehouse-Scale Computers，WSCs)
  - 单数据中心数十万处理器核心
  - 目前以X86桌面/服务器处理器为主
  - 集成网络和应用架构，通常采用虚拟化技术
  - 支持各类异构计算：GPU, FPGA, ASIC (TPU, Blockchain)
- 机器学习：尤其是深度学习
  - 专用的深度学习加速硬件
  - 两类不同的应用：训练（Training）及推理（Inference）

这是最好的时代，也是最坏的时代



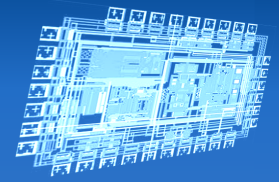
Yale Patts的技术突变三大要素

- 新的需求
  - 人工智能，尤其是深度学习的计算需求
  - 移动设备，AR/VR
  - 云计算
- 技术瓶颈
  - Memory Wall
  - Power Wall
  - ILP Wall
- 运气与机遇
  - 新材料？新架构？

ILP

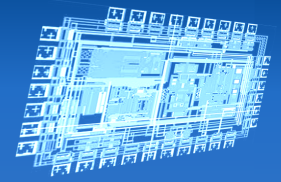
ILP: Instruction-Level Parallelism是指给定计算机代码中有多少指令可以同时并行执行。

体系结构的黄金时代?



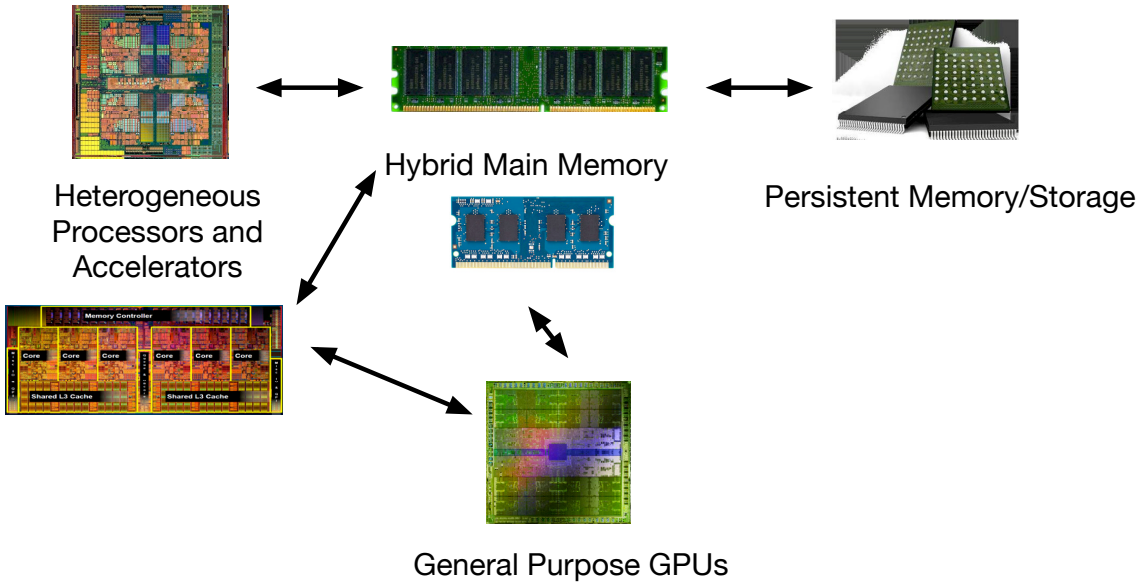
- 工业界的硬件架构设计理念已经开始逐步转变
  - 单一桌面处理器称霸的情况转变为百家争鸣百花齐放的局面
- 新的架构设计理念下出现了大量新的问题
  - 功耗问题
  - 设计复杂度及可靠性
  - 内存瓶颈
  - 编程难度
  - 硬件设计周期
- 大量问题目前还没有可靠公认的解决方案，需要大量探索工作

计算机体系结构未来的发展方向

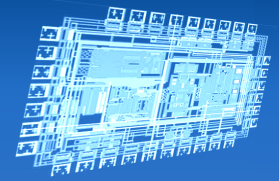


图灵奖演讲中的观点

- 领域特定的语言及架构（Domain specific architecture/languages）
  例如：深度学习系统的加速
- 开放架构（Open architectures）
- 硬件敏捷开发（Agile hardware development）



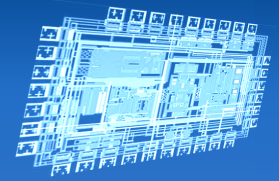
下次课程



- 指令集设计原则
- 阅读资料

  [3] H&P, Appendix A

参考文献 I



[1] W. Buchholz, “Planning a computer system: Project stretch,” 1962.

[2] G. M. Amdahl, G. A. Blaauw, and F. Brooks, “Architecture of the ibm system/360,” *IBM Journal of Research and Development*, vol. 8, no. 2, pp. 87–101, 1964.

[3] J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach, 6th ED*. Elsevier, 2011.

[4] J. P. Shen and M. H. Lipasti, *Modern processor design: fundamentals of superscalar processors*. Waveland Press, 2013.

[5] D. A. Patterson and J. L. Hennessy, *Computer organization and design: the hardware/software interface*. Newnes, 2013.

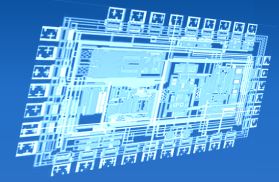
[6] Y. N. Patt, S. J. Patel, and J. Patel, “Introduction to computing systems: from bits and gates to c and beyond,” 2004.

[7] Y. Patt, “Requirements, bottlenecks, and good fortune: Agents for microprocessor evolution,” *Proceedings of the IEEE*, vol. 89, no. 11, pp. 1553–1559, 2001.

[8] J. L. Hennessy and D. A. Patterson, “A new golden age for computer architecture,” *Communications of The ACM*, vol. 62, no. 2, 2019.

[9] J. Bentley, *Programming pearls*. ACM, 1986.

参考文献 II



[10] G. E. Moore, “Cramming more components onto integrated circuits,” *Electronics*, vol. 38, no. 8, p. 144, 1965.