中山大學 SUN YAT-SEN UNIVERSITY

国家超级计算广州中心 NATIONAL SUPERCOMPUTER CENTER IN GUANGZHOU

Advanced Computer Architecture

高级计算机体系结构

第10讲：概述、ISA&ILP

张献伟

xianweiz.github.io

DCS5637, 11/2/2022

中山大學 SUN YAT-SEN UNIVERSITY

Part-I: About

Who am I?

博士，2011 – 2017，University of Pittsburgh

学士，2007 – 2011，西北工业大学

中山大學 副教授，2020.10 – 今

AMD 工程师/研究员，2017.08 – 2020.09

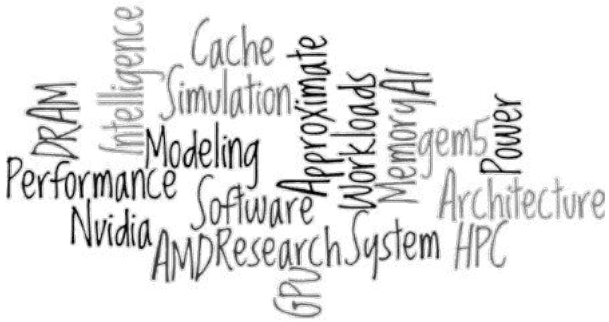
NVIDIA 实习研究员，2016.05 – 2016.08

ARCS SYSU Reines Computing System Lab

计算机体系结构

高性能及智能计算

编程及编译优化



本：编译原理/实验(21s/22s)、体系结构(22f)

研：高级计算机体系结构(21f/22f)

Example: MS Course Requirements

- The 30 credits must include one course from each of the following foundation areas.
  - 30: 24 credits + MS thesis, or 27 credits + MS project
- Foundation area courses must be completed with a grade of "B" or better.

Theory and Algorithms

- CS 2150 - DESIGN & ANALYSIS OF ALGORITHMS or
- CS 2110 - THEORY OF COMPUTATION or
- CS 2012 - ALGORITHM DESIGN or
- CS 1511 - INTRODUCTION TO THEORY OF COMPUTATION

Architecture and Compilers

- CS 2410 - COMPUTER ARCHITECTURE or
- CS 2210 - COMPILER DESIGN

Operating Systems and Networks

- CS 2510 - COMPUTER OPERATING SYSTEMS or
- CS 2520 - WIDE AREA NETWORKS

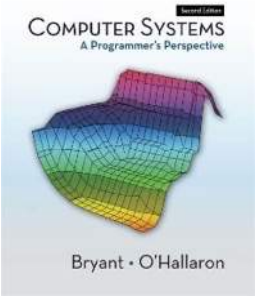
Artificial Intelligence and Database Systems

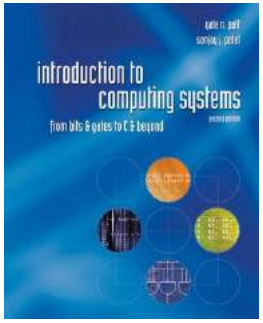
- CS 2710 - FOUNDATIONS OF ARTIFICIAL INTELLIGENCE or
- CS 2550 - PRINCIPLES OF DATABASE SYSTEMS

https://www.cs.pitt.edu/current-students/ms-computer-science/degree-requirements

Textbook[课程教材]

- 主要教材
  - John L. Hennessy and David A. Patterson，计算机体系结构：量化研究方法（英文版·原书第6版）
    - 非必须购买
- 参考资料
  - ECE 447, Onur Mutlu (Carnegie Mellon U.)
  - CIS 501, Milo Martin (U Penn)
  - Computer Organization and Design RISC-V Edition: The Hardware Software Interface (2nd Edition), Hennessy and Patterson
  - Computer Systems: A Programmer's Perspective (CSAPP), Bryant and O'Hallaron
  - Introduction to Computing Systems: From Bits and Gates to C and Beyond, Patt and Patel





Textbook[课程教材] (cont.)

John Hennessy and David Patterson circa 1991, upon publication of their groundbreaking book on computer architecture. (Shane Harvey photo)

Pioneers of Modern Computer Architecture Receive ACM A.M. Turing Award

Hennessy and Patterson's Foundational Contributions to Today's Microprocessors Helped Usher in Mobile and IoT Revolutions

Groundbreaking Textbook

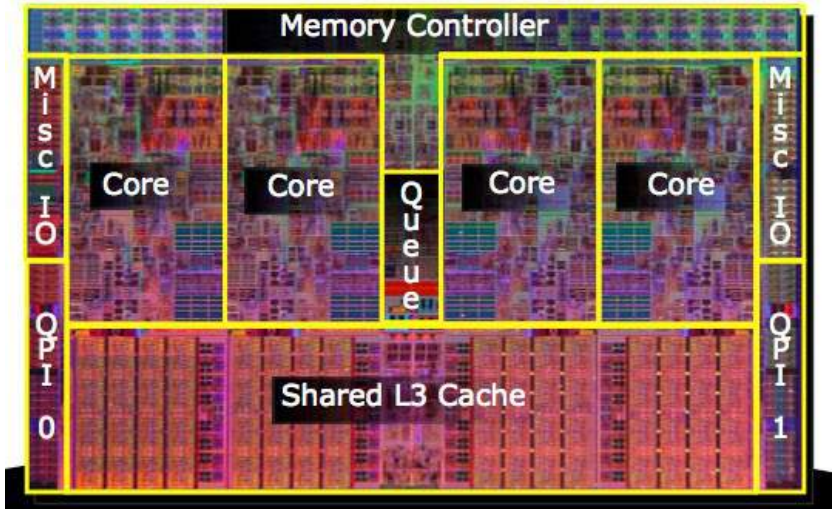
Hennessy and Patterson presented new scientifically-based methodologies in their 1990 textbook *Computer Architecture: a Quantitative Approach.* The book has influenced generations of engineers and, through its dissemination of key ideas to the computer architecture community, is credited with significantly increasing the pace of advances in microprocessor design. In *Computer Architecture*, Hennessy and Patterson encouraged architects to carefully optimize their systems to allow for the differing costs of memory and computation. Their work also enabled a shift from seeking raw performance to designing architectures that take into account issues such as energy usage, heat dissipation, and off-chip communication. The book was groundbreaking in that it was the first text of its kind to provide an analytical and scientific framework, as well as methodologies and evaluation tools for engineers and designers to evaluate the net value of microprocessor design.

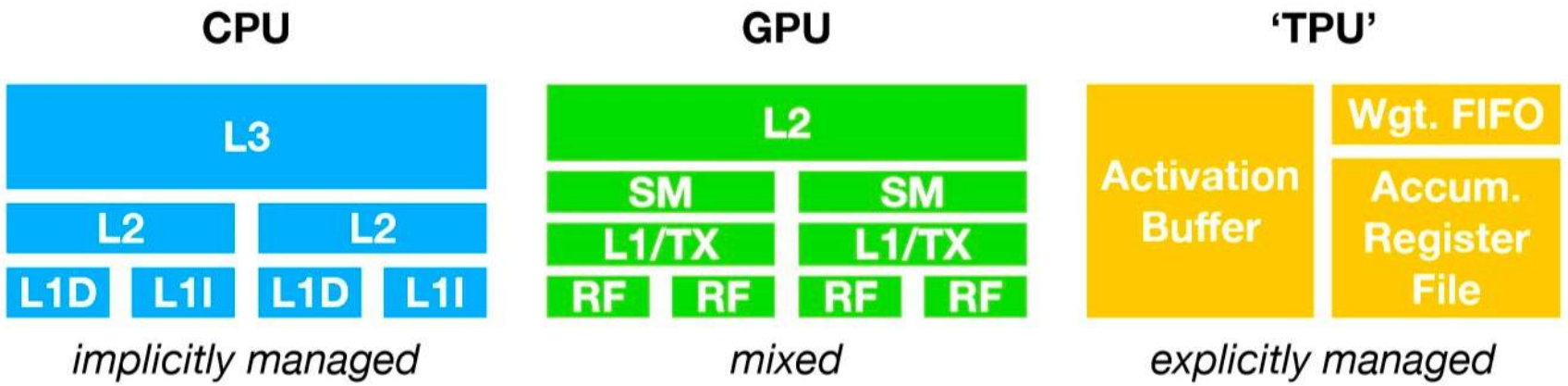
https://www.acm.org/media-center/2018/march/turing-award-2017

Outline[内容安排]

- Overview and Fundamentals[概览与基础]
- Instruction Set Architecture[指令集架构]
  - Quick intro/review
- Instruction Level Parallelism[指令级并行]
  - Pipelining, Branch Prediction, Instruction Scheduling
- Memory Hierarchy[存储层级]
  - Memory, Cache, Virtual Memory
- Data/Thread-Level Parallelism[数据/线程级并行]
  - SIMD, GPU
- Domain-specific Architectures[领域专用架构]
- And more …

Examples


Memory Controller
Misc IO
Core
Core
Queue
Core
Core
Misc IO
QPI 0
Shared L3 Cache
QPI 1


CPU
L3
L2
L2
L1D
L1I
L1D
L1I
implicitly managed
GPU
L2
SM
SM
L1/TX
L1/TX
RF
RF
RF
RF
mixed
'TPU'
Activation Buffer
Wgt. FIFO
Accum. Register File
explicitly managed

Architect[架构和架构师]

- Computer architect[计算机架构师]
  - To make design trade-offs across the hw/sw interface to meet functional, performance and cost requirements
- Being an architect is not easy[并不容易]
  - You need to consider many things in designing a new system + have good intuition/insight into ideas/tradeoffs
- But, it is fun and can be very technically rewarding[会很有意思]
- And, enables a great future[影响未来]
  - Advancement of computer architecture is vital to all other areas of computing
    - E.g., IoT, Embedded, Mobile, Data centers, HPC

Role of [Computer] Architect[职责]

- Look backward (to the past)
  - Understand tradeoffs and designs, upsides/downsides, past workloads. Analyze and evaluate the past.
- Look forward (to the future)
  - Be the dreamer and create new designs. Listen to dreamers.
  - Push the state of the art. Evaluate new design choices.
- Look up (towards problems in the computing stack)
  - Understand important problems and their nature.
  - Develop architectures and ideas to solve important problems.
- Look down (towards device/circuit technology)
  - Understand the capabilities of the underlying technology.
  - Predict and adapt to the future of technology (you are designing for *N* years ahead). Enable the future technology.

Why Study CA?[为什么学习体系结构]

- Understand why computers work the way they do
  - Source code --> instructions --> executions
  - Processors --> cache/memory --> storage
- Understand where computers are going
  - Future capabilities drive the (computing) world
  - Real-world impact: no computer architecture -> no computers!
- Understand high-level design concepts and performance
  - The best architects understand all the levels (hw, OS, apps, alg)
  - Need to understand hardware to write fast software
- Job?
  - Your MS/PhD research may involve CA
  - Your job positions (design or research, hw or sw) may need CA
  - You may manage a team working on systems

Positions???[工作职位]

CAREERS AT NVIDIA

Deep Learning Architect – New College Grad

US, CA, Santa Clara

Apply

NVIDIA is seeking computer architects to help design processor and system architectures that will enable compelling Deep Learning performance, architecture and efficiency improvements. This role offers the opportunity to directly impact the future hardware roadmap in a fast-growing technology company that leads the AI revolution. If you are obsessed with improving deep learning performance beyond anything possible with today's hardware and software, this is the place to be.

What you'll be doing:

- Understand, analyze, profile, and optimize deep learning training workloads on state-of-the-art hardware and software platforms.
- Guide development of future generations of deep learning processors and computing platforms.
- Develop detailed performance models and simulators for computing systems accelerating DL training.
- Collaborate across the company to guide the direction of machine learning at NVIDIA; spanning teams from hardware to software and research to production.
- Drive HW/SW co-design of NVIDIA's full deep learning platform stack, from silicon to DL frameworks.

What we'd like to see:

- You are pursuing a PhD or MS or have equivalent in CS, EE or CSEE (or equivalent experience).
- Strong background in computer architecture, preferably with focus on high-performance parallel processors.
- Background in machine learning and neural networks, in particular training.
- Experience analyzing and tuning application performance.
- Experience with processor and system-level performance modelling.
- Programming skills in C++ and Python.
- Familiarity with GPU computing (CUDA, OpenCL).

达摩院-芯片性能架构师-北京

发布时间：	2021-04-20	工作地点：	北京	工作年限：	五年以上
所属部门：	阿里集团	学　历：	硕士	招聘人数：	若干

团队介绍：

阿里巴巴集团达摩院旗下的数据计算-计算技术实验室致力于前瞻性研究，探索异构计算，存储，和互联的系统架构，软硬件协同设计，体系结构，电路设计，编译和编程环境等方面的技术问题，研制高性能、低功耗的异构计算系统，人工智能计算芯片，以及其他芯片架构及系统。通过自上而下基于应用驱动和自下而上基于新技术的研究方法，利用计算机体系结构设计优化，VLSI等领域的技术积累与合作伙伴在计算资源优化、新计算体系方向等构建创新系统，提升阿里巴巴集团计算能力并复用于国民经济的各行各业中。

岗位描述：

参與新型計算芯片的開發，作用包括：
1.分析業務軟件的性能特徵，在系統和仿真環境中分析性能如何，以幫助芯片架構的設計決策。
2.開發和維護性能分析工具，實現數據分析
3.參與SOC模塊的性能建模，和設計團隊一起優化芯片的性能，並和翻積。
4.開發性能測試用例，測試系統的微架構特徵和設計參數。
5.在新興計算中。領域（機器學習，視頻計算等），分析業務特點，推動軟硬件協同設計。

岗位要求：

1.熟愛芯片架構，願意學習。
2.軟件編程能力（如C ++，python）
3.具有計算機體系結構背景，熟悉SOC組件。
職位加分：
1.有相關的計算機體系結構研究經歷。
2.有性能模型建模的經驗。

Golden Age for CA[黄金时代]

- Today is a very exciting time to study architecture
  - Many new demands from the top
  - Fast changing demands and personalities of users
  - Many new issues at the bottom
- Computing landscape is very different from 10-20 years ago (Recall: Intel = 25*Nvidia → 0.34*Nvidia)
  - Every component and its interfaces, as well as entire system designs are being re-examined
  - You can revolutionize the way computers are built, if you understand both the hardware and the software (and change each accordingly)
- **No clear, definitive answers to these problems[有问题，缺方案]**

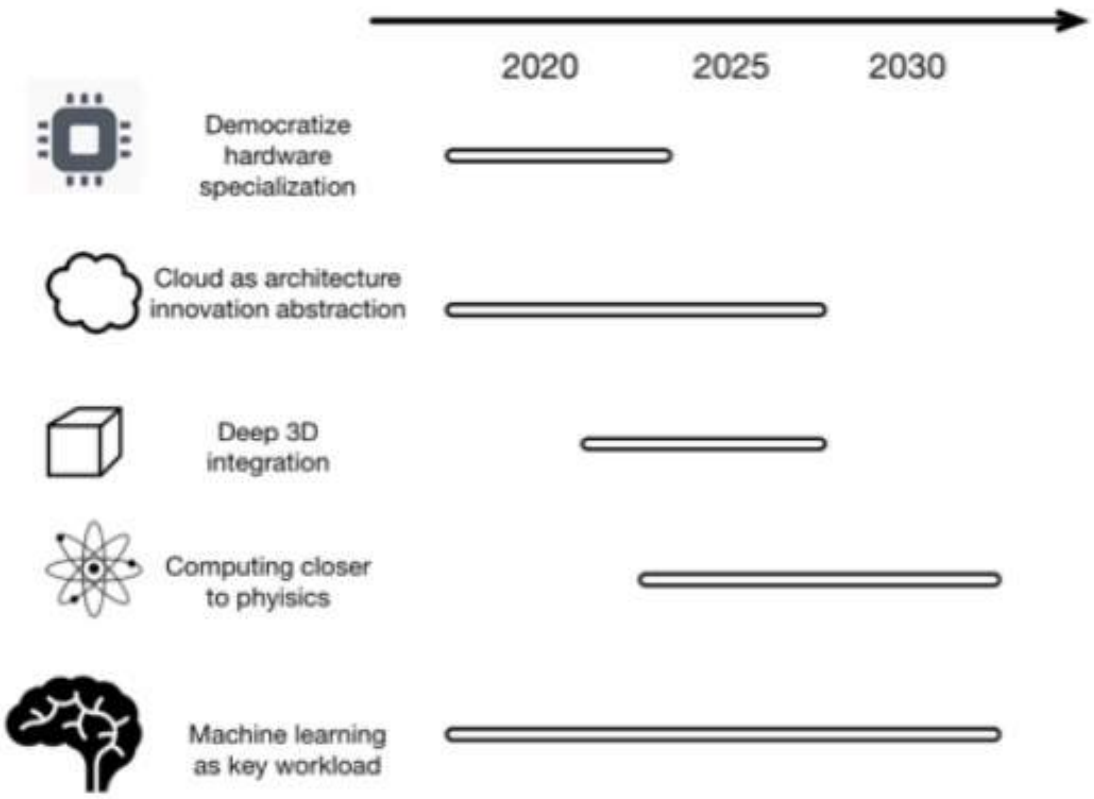
Industry[工业界]

- Nvidia/AMD, 08/2022: US bans Nvidia and AMD GPU chips to China
- Intel, 08/2022: Kills Optane Memory[2]
- Apple, 11/2020: M1, ARM-based SoC[3]
- AMD, 10/2020: Acquire Xilinx[4]
- Intel, 09/2020: Xe GPU[5]
- Samsung, 11/2019: Cease CPU development[6]
- Amazon, 11/2018: AWS Graviton[7]
- Intel/IBM/ARM, 01/2018: Meltdown and Spectre[8]
- Micron, 03/2021: Cease 3D-XPoint, invest CXL[9]
- AI Chips: Graphcore, Habana Labs, Cerebras, Cambricon…

[1] https://www.sec.gov/ix?doc=/Archives/edgar/data/1045810/000104581022000146/nvda-20220826.htm
[2] https://www.tomshardware.com/news/intel-kills-optane-memory-business-for-good
[3] https://pdf.wondershare.com/macos/everything-about-apple-m1-chip.html
[4] https://www.amd.com/en/corporate/xilinx-acquisition
[5] https://www.intel.com/content/www/us/en/products/discrete-gpus/iris-xe-aic.html
[6] https://www.anandtech.com/show/15061/samsung-to-cease-custom-cpu-development
[7] https://aws.amazon.com/ec2/graviton/
[8] https://people.redhat.com/pladd/Meltdown-Spectre-NYRHUG-2018-01.pdf
[9] https://investors.micron.com/news-releases/news-release-details/micron-updates-data-center-portfolio-strategy-address-growing

Academia[学术界]

Architecture 2030 Workshop @ ISCA 2016



John L. Hennessy, David A. Patterson

- Current challenges[问题]
  - End of Moore's Law and Dennard Scaling
  - Overlooked security
- Future opportunities in computer architecture[机遇]
  - Domain-specific architectures
  - Domain-specific languages
  - Open architectures
  - Agile hardware development

[1] Arch2030, https://arxiv.org/pdf/1612.03182.pdf (2016)
[2] A New Golden Age for Computer Architecture (2019)

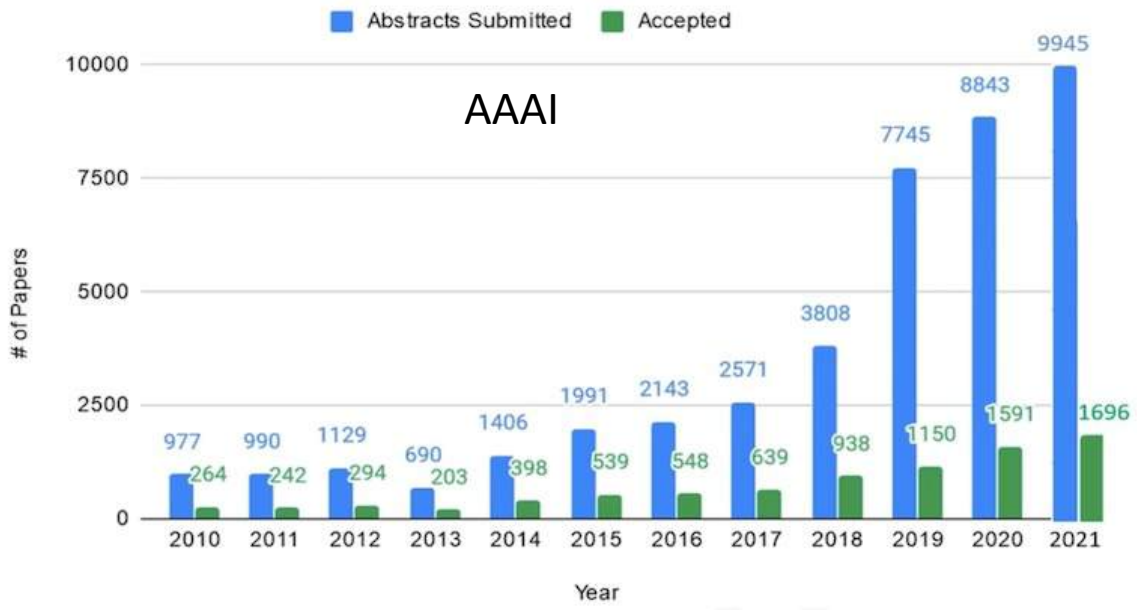
Top-tier Conferences[顶级会议]

- **ISCA**
  - The International Symposium on Computer Architecture (ISCA)
  - 2021: 48th, 76/407 papers (18.6% acceptance rate)
- **MICRO**
  - The IEEE/ACM International Symposium on Microarchitecture
  - 2021: 54th, 94/430 papers (21.8% acceptance rate)
- **HPCA**
  - IEEE International Symposium on High-Performance Computer Architecture
  - 2021: 27th, 63/258 papers (24.4% acceptance rate)
- **ASPLOS**
  - ACM International Conference on Architectural Support for Programming Languages and Operating Systems
  - 2021: 26th, 75/398 (18.8% acceptance rate)

Arch vs. AI

Year	Submitted	Accepted	Rate
ISCA '19	365	62	17%
ISCA '17	322	54	17%
ISCA '13	288	56	19%
ISCA '12	262	47	18%
ISCA '11	208	40	19%
ISCA '10	245	44	18%

Year	Submitted	Accepted	Rate
ASPLOS '19	351	74	21%
ASPLOS '18	319	56	18%
ASPLOS '17	320	53	17%
ASPLOS '16	232	53	23%
ASPLOS '15	287	48	17%
ASPLOS '14	217	49	23%



HPCA'2022 (High-Performance Computer Architecture)

- Sessions
  - Accelerators (4)
  - Security (3)
  - Quantum (3)
  - At Scale
  - Storage, Scheduling, Interfaces
  - Simulation
  - Cache Hierarchy
  - Synthesis
  - Traditional Architecture

[1] https://hpca-conf.org/2022/program

ISCA'2022 (Computer Architecture)

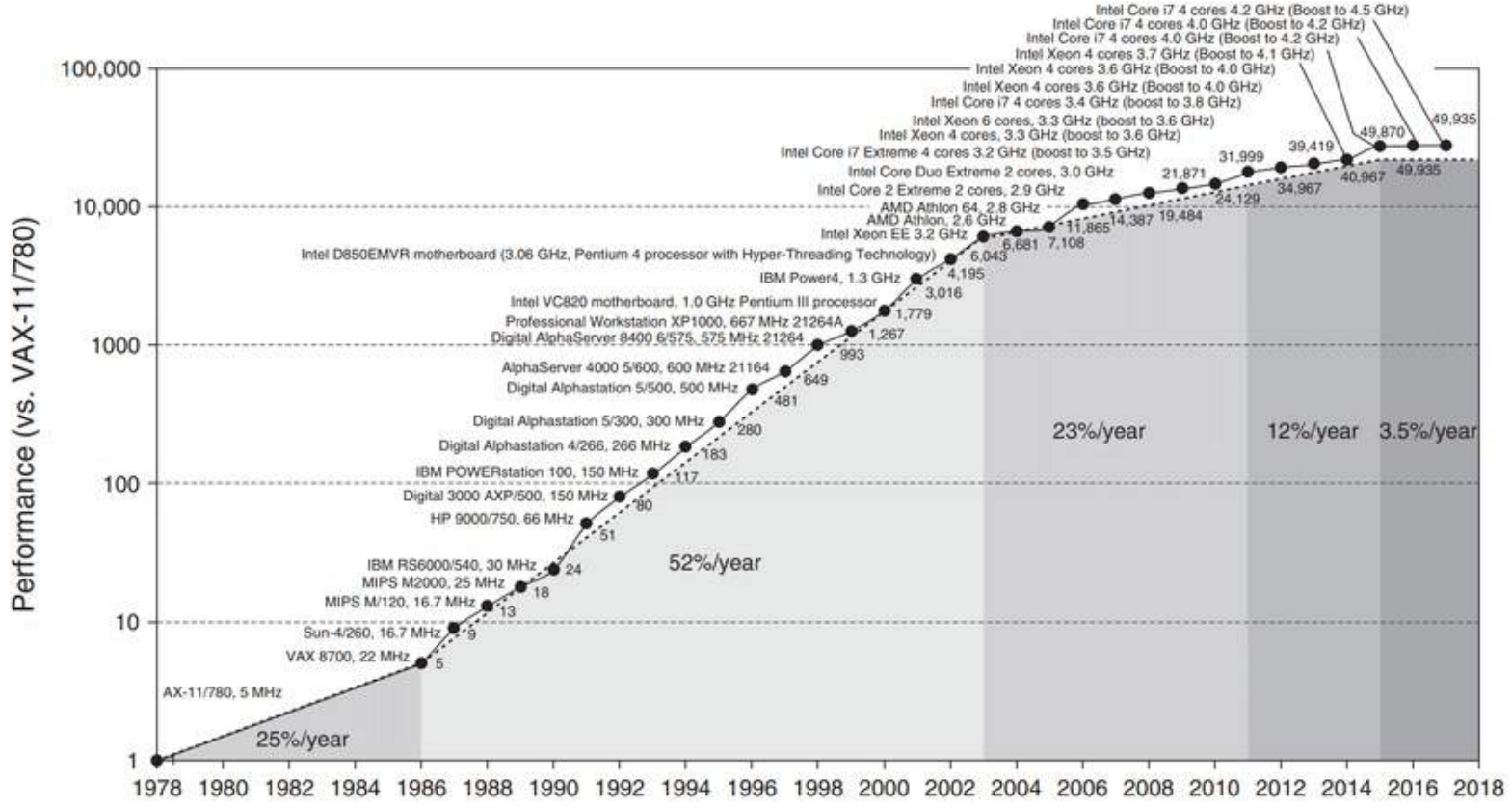
- Sessions
  - Security (4)
  - Graph Applications
  - Processing in Memory
  - Industry Session
  - Persistent Memory
  - Quantum
  - Microarchitecture and Parallelism
  - Novel Architectures (3)
  - Learning (2)

[1] https://www.iscaconf.org/isca2022/program/

Part-II: Introduction

Technology Improvement[技术提升]

- Computer technology has greatly improved
  - A $500 cellphone today outperforms the fastest supercomputer in 1993 ($50 million)
  - Improvement from both advances in the tech used to build computers and from innovations in computer design

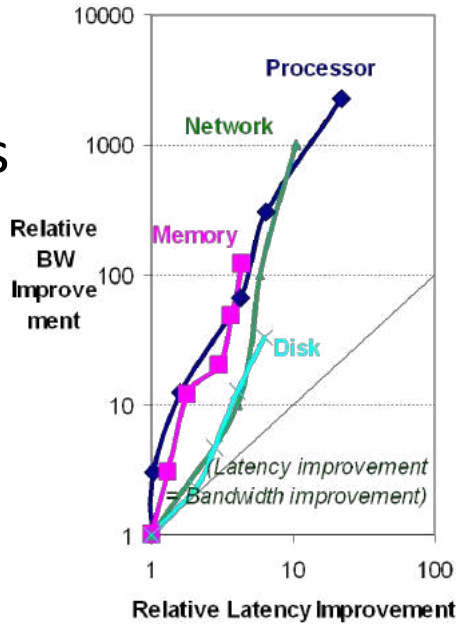


Trends in Technology (§1.4)[技术趋势]

- Integrated circuit (IC) logic[集成电路]
  - Transistor density: +35%/year (feature size decreases)
  - Die size: +10-20%/year
  - Integration overall: +40-55%/year (Moore's Law)
- DRAM capacity: +25-40%/year (growth is slowing)[内存]
  - Memory usage quadruples every three years
- Flash capacity: +50-60%/year[闪存]
  - 8-10X cheaper/bit than DRAM
- Magnetic disk: +40%/year[磁盘]
  - 8-10X cheaper/bit then Flash and 200-300X cheaper/bit than DRAM
- Network[网络]

Performance Trends[性能趋势]

- Bandwidth or throughput[带宽/吞吐]
  - Total work done in a given time
  - 32,000 - 40,000X improvement for processors
  - 400 - 2400X improvement for memory and disks
- Latency or response time[时延/响应时间]
  - Time between start and completion of an event
  - 50 - 90X improvement for processors
  - 8 - 9X improvement for memory and disks
    - Memory wall[内存墙]



- Latency lags bandwidth (in the last 30 years)

Transistors and Wire[晶体管/线路]

- IC processes are characterized by feature size[特征尺寸]
- **Feature size**: minimum size of transistor or wire
  - 10um in 1971 to 22nm in 2012 to 7nm in 2017 to 5nm in 2020 (3nm is being developed)
- Moore's Law: aka "technology scaling"[缩放]
  - Literally: density (transistors/area) doubles every 18 months
  - Public interpretation: performance doubles every 18 months
  - Continued miniaturization (esp. reduction in channel length)
    - + Improves switching speed, power/transistor, area(cost)/transistor
    - – Reduces transistor reliability

Intel 7: 10nm

Intel 4: 7nm

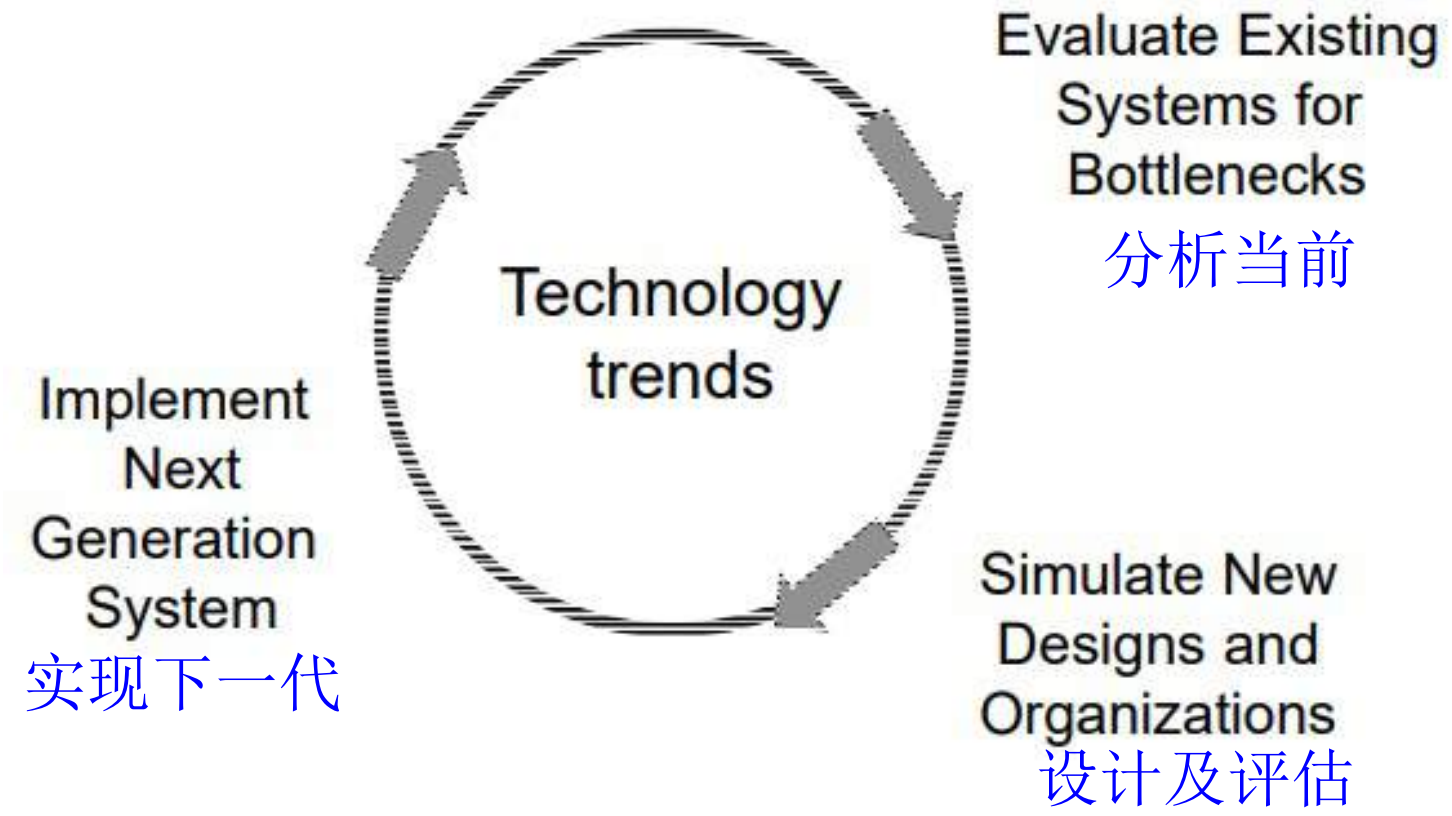
Intel: 7+

Intel 20A: 5nm

Power and Energy (§1.5)[功耗/能耗]

- **Energy** is a biggest challenge facing computer design
  - Bring power in with 100s of pins
  - Power is dissipated as heat and must be removed
- **Power/energy** are increasingly important
  - Battery life for mobile devices[电池续航]
    - Laptops, phones, cameras
  - Tolerable temperature for devices without active cooling[温度]
    - Power means temperature, active cooling means cost
    - No room for a fan in a cell phone, no market for a hot cell phone
  - Electric bill for compute/data centers[电费]
    - Pay for power twice: once in, once out (to cool)
  - Environmental concerns[环保]
    - “Computers” account for growing fraction of energy consumption

Methodology: Design/Evaluation[方法]



Design Goals[设计目标]

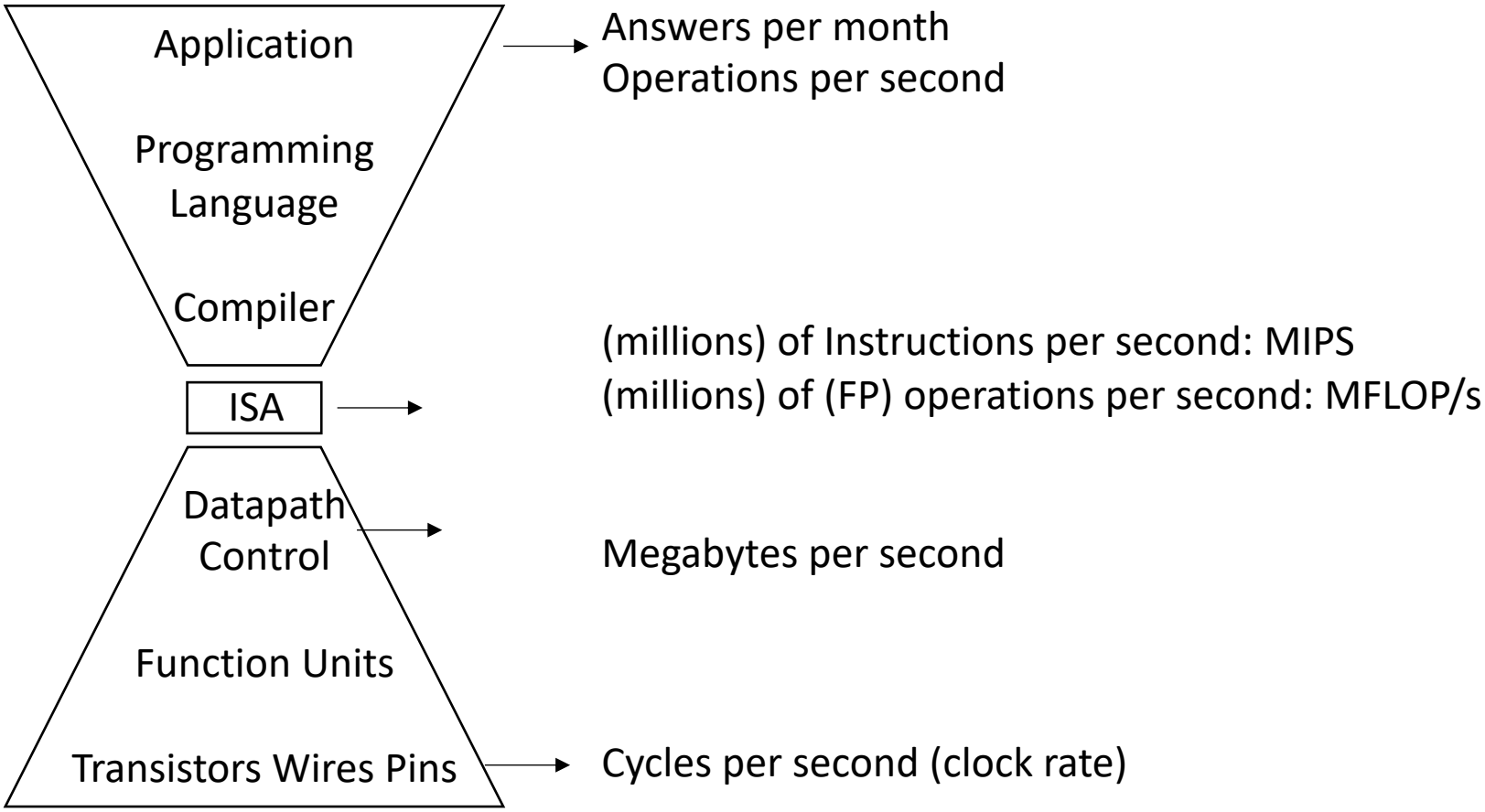
- Functional[功能性]
  - What functions should it support？
  - Needs to be correct
    - Unlike software, difficult to update once deployed
- High performance[高性能]
  - "Fast" is only meaningful in the context of a set of important tasks
  - Not just "Gigahertz"
  - Impossible goal: fastest possible design for all programs
- Reliable[可靠性]
  - Does it continue to perform correctly?
  - Hard fault vs. transient fault
    - Example： memory errors and sun spots
  - Space satellites vs. desktop vs. server reliability

Design Goals (cont.)

- Low cost[低成本]
  - Design cost (huge design teams, why?)[设计]
  - Cost of making first chip after design (mask cost)[流片]
  - Per unit manufacturing cost (wafer cost)[量产]
- Low power/energy[低能耗]
  - Energy in (battery life, cost of electricity)
  - Energy out (cooling and related costs)
  - Cyclic problem, very much a problem today
- **Challenge**: balancing the relative importance of these goals
  - And the balance is constantly changing
  - No goal is absolutely important at expense of all others
  - Our focus: performance, only touch on cost, power, reliability

Performance (§1.8)[性能]

- The performance metric may mean different things

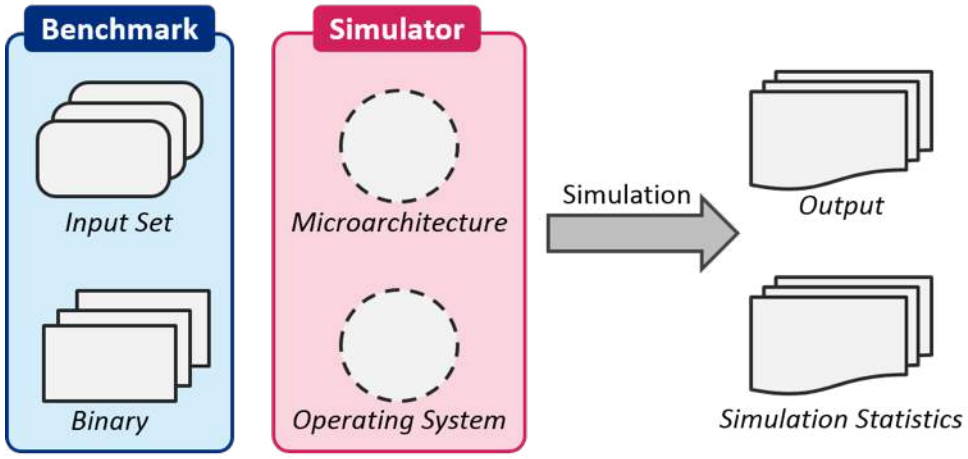


Measuring Performance[评估性能]

- Time to run the task (latency)
  - Execution time, response time, CPU time, …
- Tasks per day, hour, week, sec, ns, …
  - Throughput, bandwidth
- Performance measurement[测试]
  - Hardware prototypes : Cost, delay, area, power estimation
  - Simulation (many levels, ISA, RT, Gate, Circuit, …)
  - Benchmarks (Kernels, toy programs, synthetic), Traces, Mixes
  - Analytical modeling and Queuing Theory

Simulator[模拟器]

- What is an architecture (or architectural) simulator?
  - A tool that reproduces the behavior of a computing device
- Why use a simulator?
  - Leverage faster, more flexible software development cycle
  - Permits more design space exploration
  - Facilitates validation before hardware becomes available
  - Possible to increase/improve system instrumentation

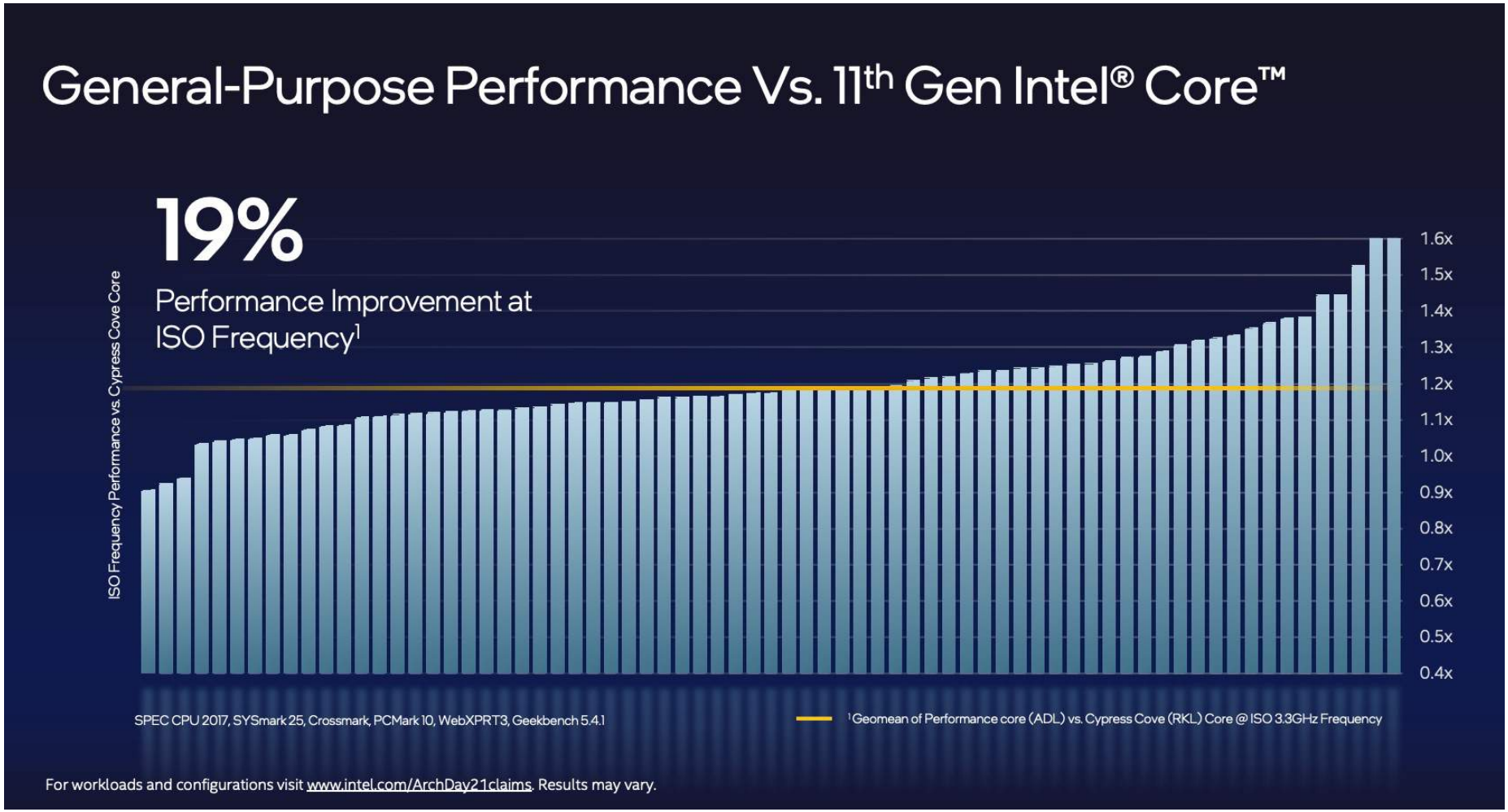


Benchmarks[基准测试集]

- SPEC: Standard Performance Evaluation Corporation
- PARSEC: Princeton Application Repository for Shared Memory Computers
- Rodinia: GPGPU applications
- HPL: a High-Performance Linpack benchmark implementation
- MLPerf: a suite of performance benchmarks that cover a range of leading AI workloads widely in use
- MediaBench: Multimedia and embedded applications
- Transaction processing: TPC-C, SPECjbb
- EEMBC: embedded microprocessor benchmark consortium

Benchmarks (cont.)

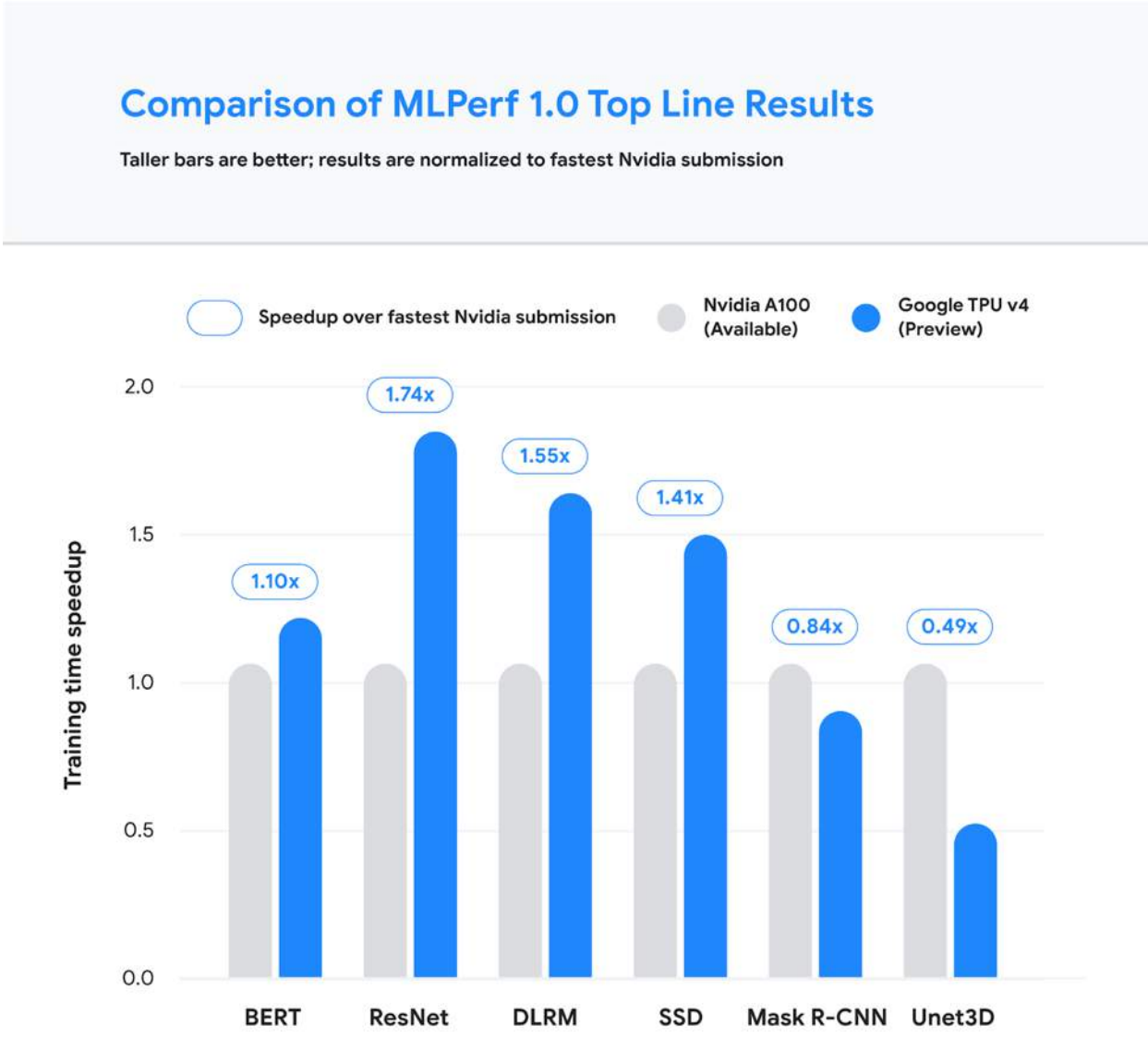
- Example: performance of Intel's newest CPU (2021)



https://download.intel.com/newsroom/2021/client-computing/intel-architecture-day-2021-presentation.pdf

Benchmarks (cont.)

- MLPerf
  - A broad ML benchmark suite for measuring performance of ML software frameworks, ML hardware accelerators, and ML cloud platforms



MLPerf

Simulation Goals Vary[不同目标]

- Explore the design space quickly and see what you want to
  - potentially implement in a next-generation platform
  - propose as the next big idea to advance the state of the art
  - the goal is mainly to see relative effects of design decisions
- Match the behavior of an existing system so that you can
  - debug and verify it at cycle-level accuracy
  - propose small tweaks to the design that can make a difference in performance or energy
  - the goal is very high accuracy
- Other goals in-between:
  - Refine the explored design space without going into a full detailed, cycle-accurate design
  - Gain confidence in your design decisions made by higher-level design space exploration

Tradeoffs in Simulation[平衡]

- Three metrics to evaluate a simulator
  - Speed, Flexibility, Accuracy
- **Speed**[速度]: How fast the simulator runs (xIPS, xCPS, slowdown)
- **Flexibility**[灵活性]: How quickly one can modify the simulator to evaluate different algorithms and design choices?
- **Accuracy**[准确度]: How accurate the performance (energy) numbers the simulator generates are vs. a real design (Simulation error)
- The relative importance of these metrics varies depending on where you are in the design process (what your goal is)

Tradeoffs in Simulation (cont.)

- Speed & flexibility affect:
  - How quickly you can make design tradeoffs
- Accuracy affects:
  - How good your design tradeoffs may end up being
  - How fast you can build your simulator (simulator design time)
- Flexibility also affects:
  - How much human effort you need to spend modifying the simulator

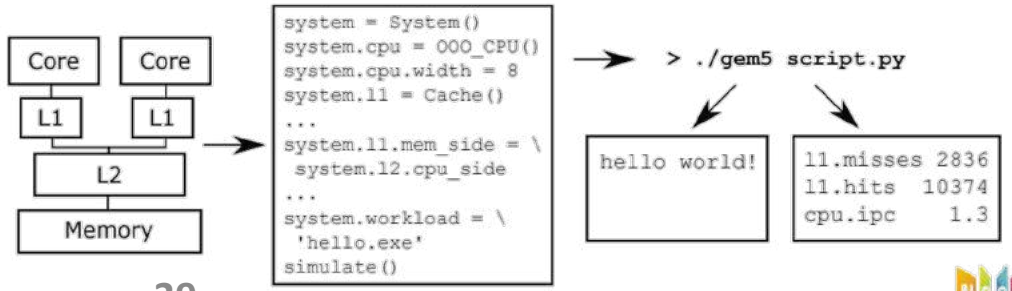
- You can trade off between the three to achieve design exploration and decision goals

High-level Simulation[高层级模拟]

- Key Idea: Raise the abstraction level of modeling to give up some accuracy to enable speed & flexibility (and quick simulator design)
  - Get first-hand insights
- Advantages
  - Can still make the right tradeoffs, and can do it quickly
  - All you need is modeling the key high-level factors, you can omit corner case conditions
  - All you need is to get the "relative trends" accurately, not exact performance numbers
- Disadvantages
  - Opens up the possibility of potentially wrong decisions
  - How do you ensure you get the "relative trends" accurately?

Example Simulator: gem5

- gem5 = Wisconsin GEMS + Michigan m5
  - The gem5 simulator is a modular platform for computer-system architecture research, encompassing system-level architecture as well as processor microarchitecture.
  - Widely used in academia and industry
- Why gem5?
  - Runs real workloads
  - Comprehensive model library (memory, IO, Full OS, Web, …)
  - Rapid early prototyping (quickly test system-level ideas)
  - Can be wired to custom models (add detail where it matters, when it matters)



https://pages.cs.wisc.edu/~sinclair/courses/cs752/fall2020/handouts/lecture/archSim.pdf

Aspects of CPU Performance

- $CPU\ time = \frac{Seconds}{program} = \frac{Instructions}{program} * \frac{Cycles}{Instructions} * \frac{Seconds}{Cycles}$

	Inst Count	CPI	Clock Rate
Program	❍		
Compiler	❍	❍	
Inst. Set	❍	❍	
Organization		❍	❍
Technology			❍

Quantitative Principles (§1.8)[量化原则]

- Guidelines and principles that are useful in the design and analysis of computers
- Take advantage of parallelism[并行]
  - System level: multiple processors, multiple disks
  - Individual processor: instruction parallelism, e.g., pipelining
  - Detailed digital design: cache, memory
- Principle of locality[局部性]
  - Programs tend to reuse data and insts they have used recently
    - A program spends 90% of its execution time in only 10% of the code
- Focus on the common case[一般情况]
  - To make a trade-off, favor the frequent case over infrequent

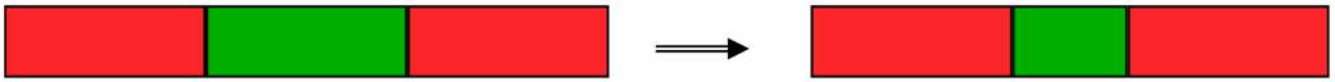
Amdahl's Law[阿姆达尔定律]

- The performance improvement to be gained from using some faster mode of execution is limited by the fraction of the time the faster mode can be used
- 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
- Amdahl's law defines the speedup that can be gained by using a particular feature
  - Speedup due to some enhancement *E*:

$$Speedup_{overall} = \frac{ExTime_{withoutE}}{ExTime_{withE}} = \frac{Performance_{E}}{Performance_{withoutE}}$$

Amdahl's Law (cont.)

- Suppose that enhancement *E* accelerates a fraction of the task by a factor *S*, and the remainder of the task is unaffected



$$ExTime_{withE}$$

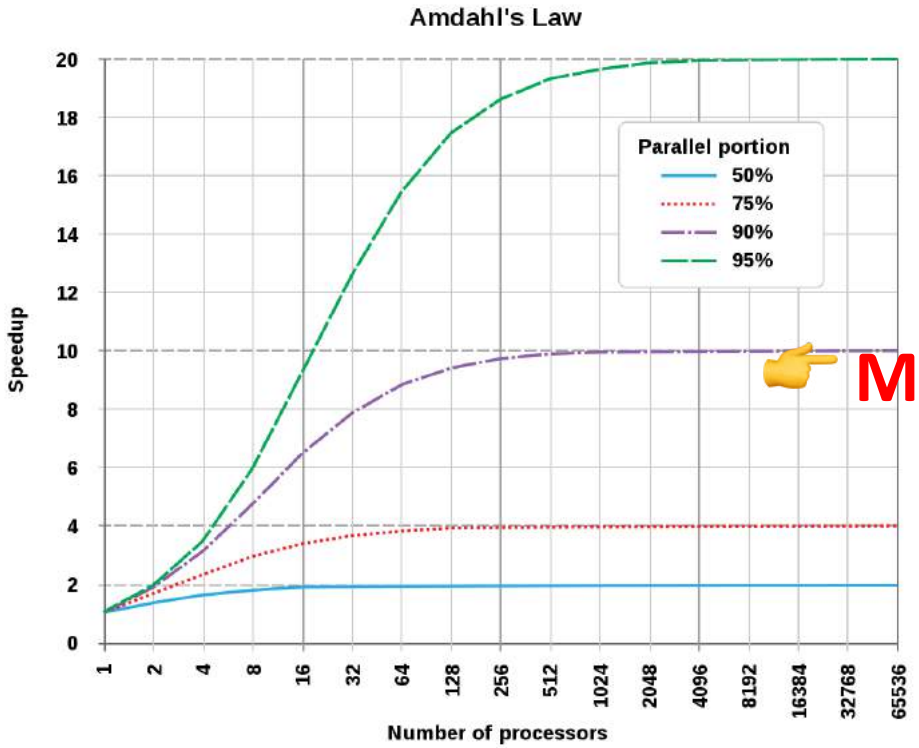
$$= ExTime_{withoutE} * [(1 - fraction_{enhanced}) + \frac{fraction_{enhanced}}{S}]$$

$$Speedup = \frac{ExTime_{withoutE}}{ExTime_{withE}}$$

$$= \frac{1}{(1 - fraction_{enhanced}) + \frac{fraction_{enhanced}}{S}}$$

Amdahl's Law (cont.)

- A program's speedup is limited by its serial part
  - For example, if 95% of the program can be parallelized, the theoretical maximum speedup using parallel computing would be **20x**

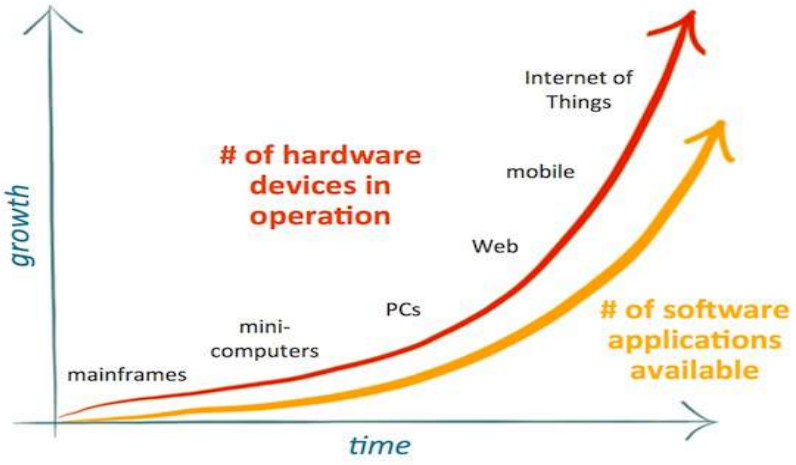


Make the fast case common !

Part-III: ISA & ILP

The History

- For more than 50 years, we have enjoyed exponentially increasing compute power[算力急剧增长]
- The growth is based on a fundamental contract between HW and SW[得益于软硬件之间的协议]
  - HW may change radically “under the hood”
    - Old SW can still run on new HW (even faster)
  - HW looks the same to SW, always speaking the same language
    - The **ISA**, allows the decoupling of SW development from HW dev



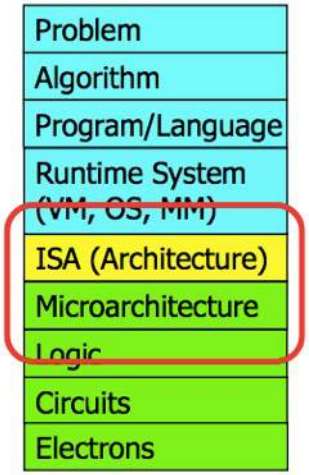
What is ISA?

- Instruction Set == A set of instructions
- The HW/SW contract[软硬件协议]
  - Compiler correctly translates source code to the ISA[编译器]
  - Assembler translates to relocatable binary[汇编器]
  - Linker solidifies relocatables into object code[连接器]
  - HW promises to do what the object code says[硬件执行]

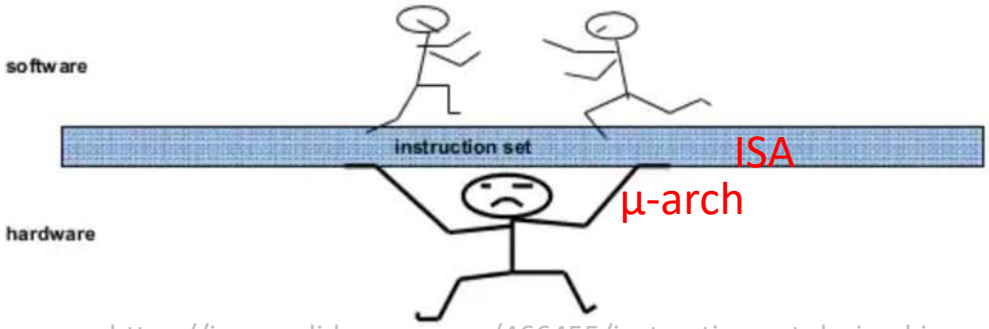
- Not in the “contract”: non-functional aspects[非协议]
  - How operations are implemented
  - Which operations are fast and which are slow and when
  - Which operations take more power and which take less

ISA + μ-arch = Arch

- "Architecture" = ISA + microarchitecture
- ISA[指令集架构]
  - Agreed upon interface between sw and hw
    - SW/compiler assumes, HW promises
  - What the software writer needs to know to write and debug system/user programs



- Microarchitecture (μ-arch)[微架构]
  - Specific implementation of an ISA
    - Implementation of the ISA under specific design constraints and goals
  - Not visible to the software



https://image.slideserve.com/466455/instruction-set-design-l.jpg