中山大學 SUN YAT-SEN UNIVERSITY

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

Computer Architecture

计 算 机 体 系 结 构

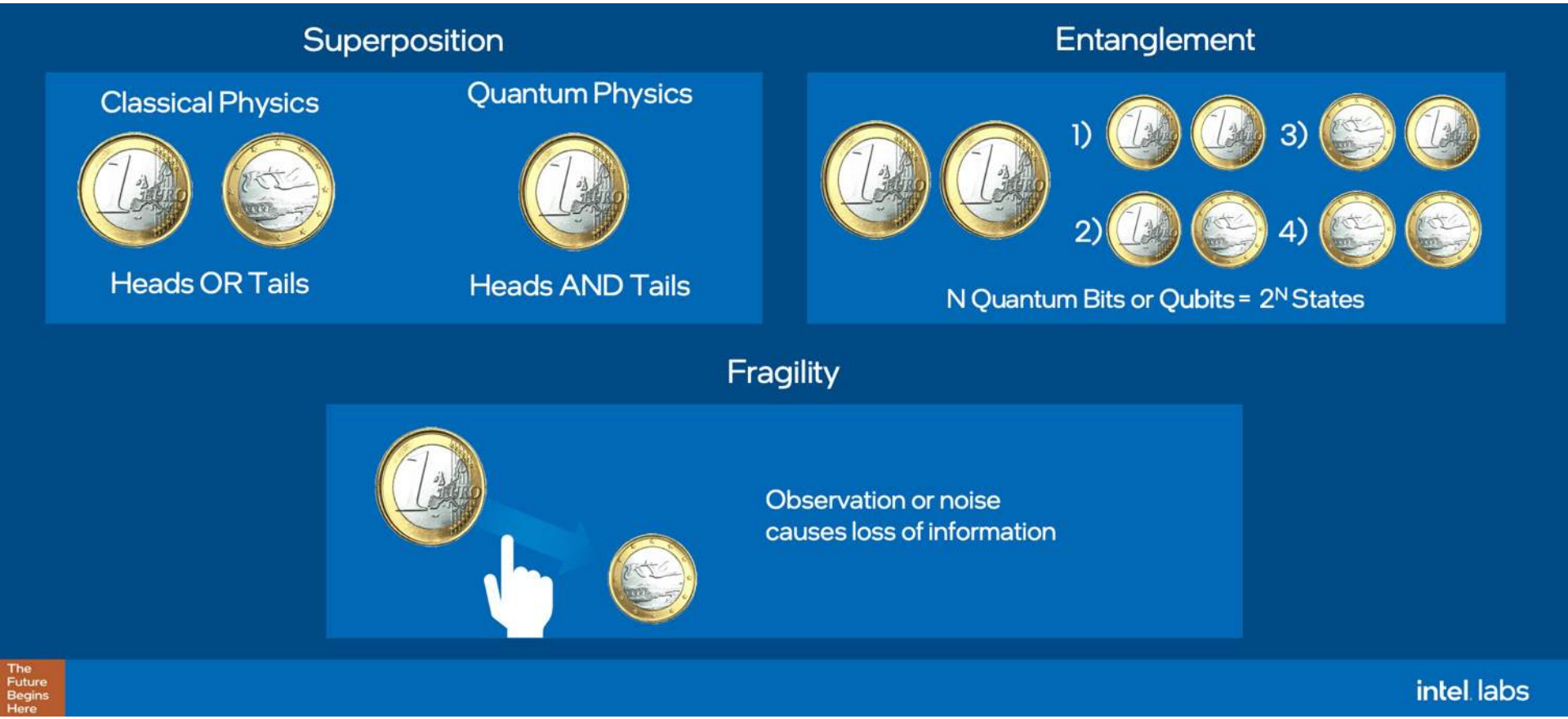
第25讲：Domain Specific Arch (4)

张献伟

xianweiz.github.io

DCS3013, 12/28/2022

Quantum Computing: Key Concepts

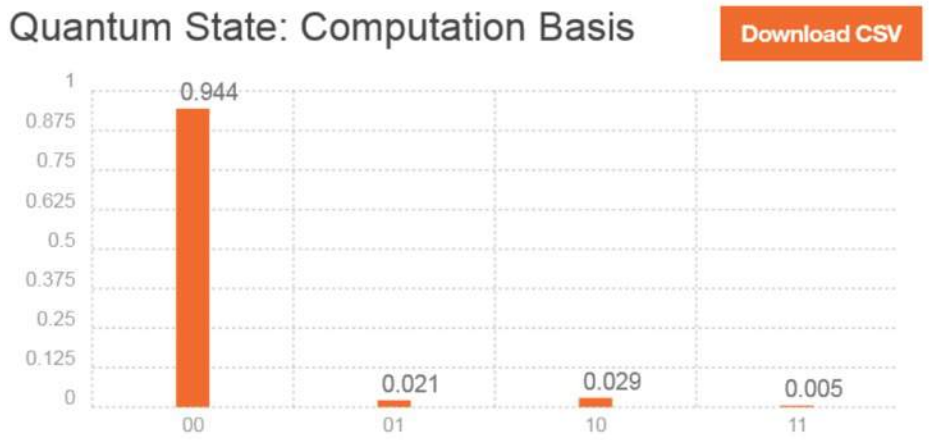


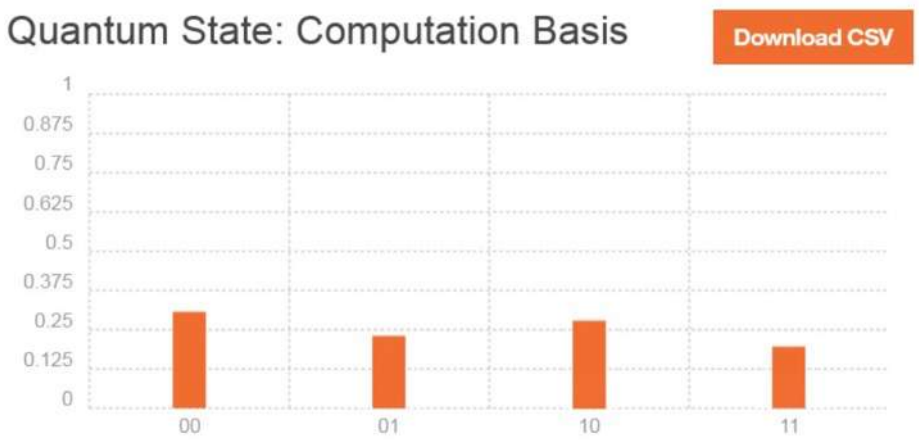
Quantum Algorithm (cont.)

- Prepare three qubits to represent all possible computational basis evenly

$$\frac{1}{2\sqrt{2}}|000\rangle + \frac{1}{2\sqrt{2}}|001\rangle + \ldots + \frac{1}{2\sqrt{2}}|111\rangle$$

- We further encode the problem information into the superposition and manipulate it
- Then we make measurements

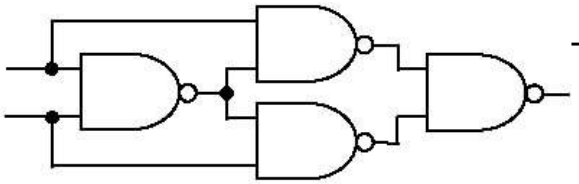




Quantum Gate[量子门]

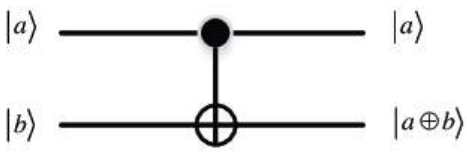
- **Quantum gate** is a transformation from one qubit state to another
  - They are the building blocks of quantum circuits, like classical logic gates are for conventional digital circuits
  - Must have the same number of inputs and outputs
    - We cannot implement a quantum operator in a quantum computer if the operation is not reversible
    - Qubits do not branch out or merge together → we don't have the nicely if-then-else or while statements
- Currently available superconducting quantum hardware only supports one-qubit gates and two-qubit gates on specific pairs
  - More complex operations must be decomposed into multiple simpler, supported ones

Exclusive OR (XOR)

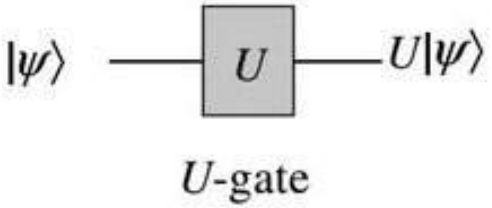


A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

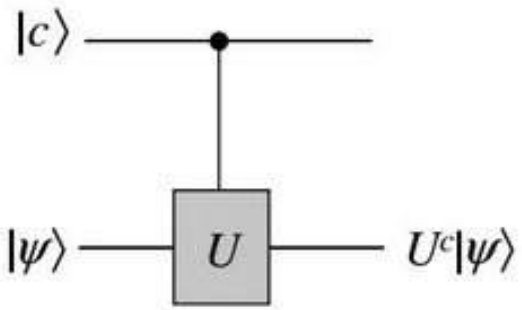
$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$



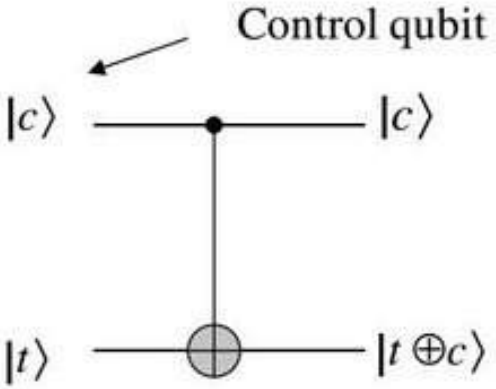
Common Quantum Gates

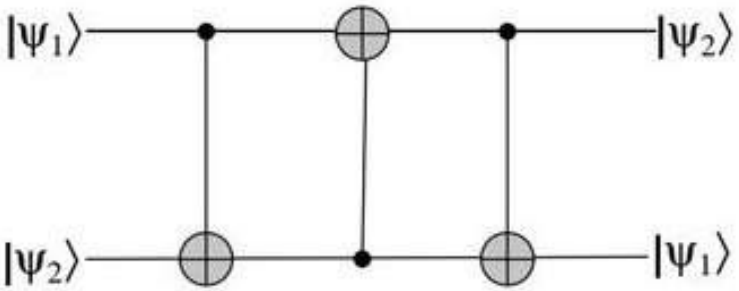


U-gate

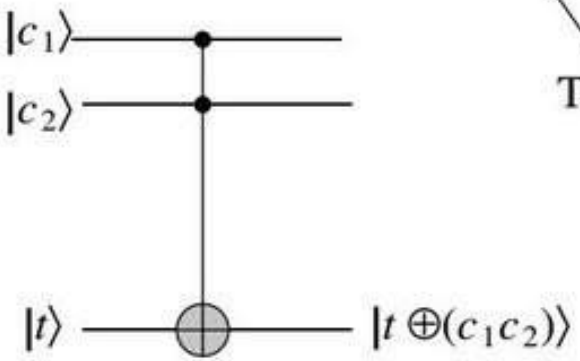


Controlled-*U* gate





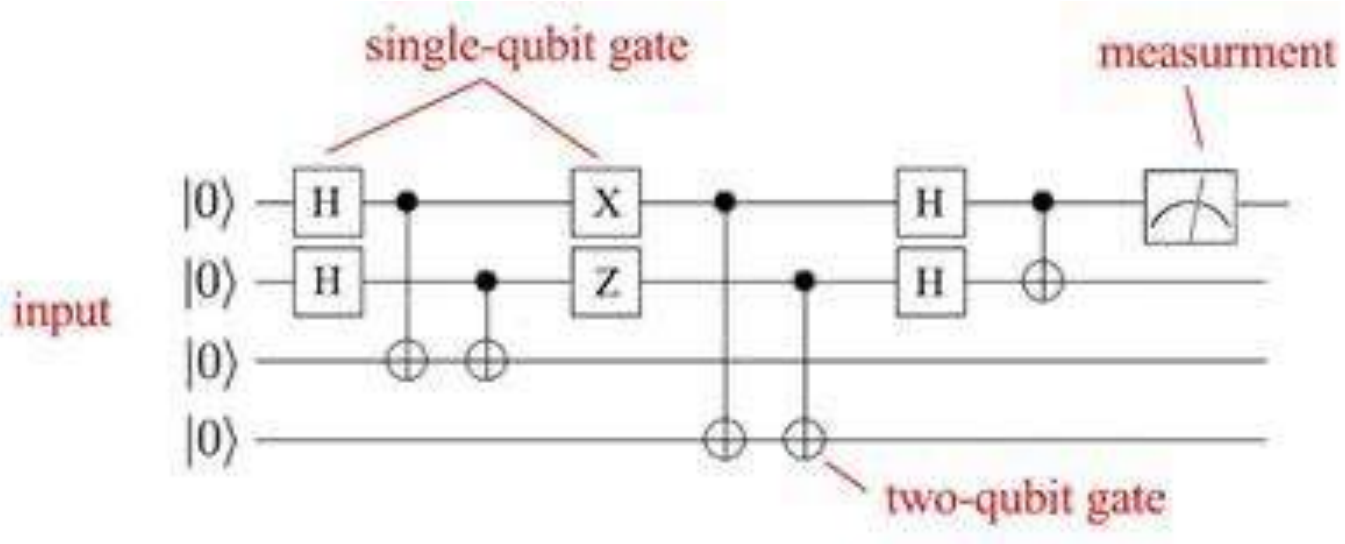
SWAP-gate



Toffoli-gate

Quantum Circuits[量子电路]

- A program can be written as a diagram with a sequence of the quantum gates (a quantum circuit)
- The majority of algorithm development is still done in terms of quantum gates
  - The quantum analog of logical operations like AND, NOT, or XOR



Quantum Program[量子程序]

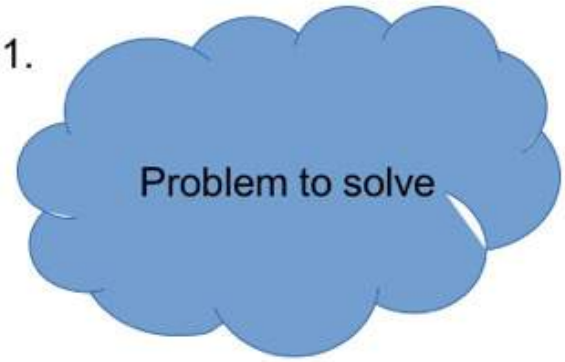
- **Quantum program**s are typically represented as a **circuit** which, like a classical program, is on ordered list of instructions
  - Here the instructions are **quantum logic gate**s applied to qubits
- Programs are usually run thousands of times to obtain a distribution over possible answers
  - Since the outcome of a quantum program is a classical bitstring
  - Since quantum systems are inherently noisy

```
[python3] $ pip install qiskit
```

```
from qiskit import QuantumProgram
qp = QuantumProgram()
qr = qp.create_quantum_register('qr',2)
cr = qp.create_classical_register('cr',2)
qc = qp.create_circuit('Bell',[qr],[cr])
qc.h(qr[0])
qc.cx(qr[0], qr[1])
qc.measure(qr[0], cr[0])
qc.measure(qr[1], cr[1])
result = qp.execute('Bell')
print(result.get_counts('Bell'))
```

Quantum Circuit Compilation[编译]

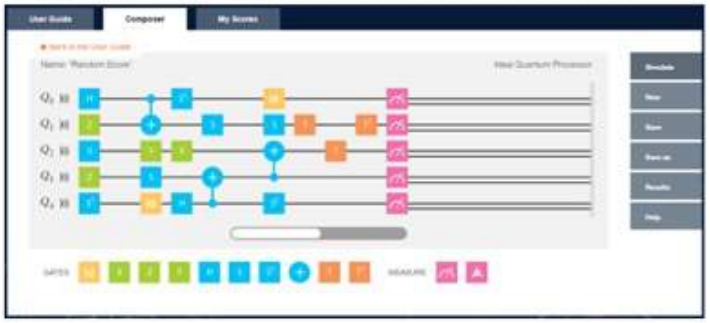
1.



* similar to compilation in classical computing (e.g. compile C++ code)

Quantum circuit for idealized hardware

2.

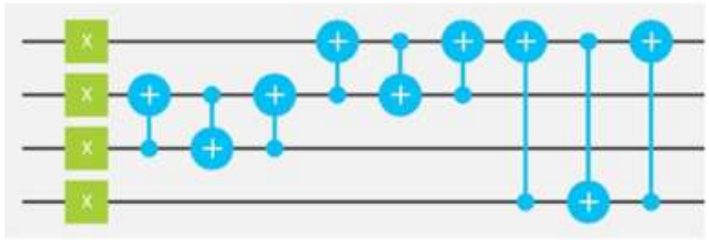


* example generated with Qiskit/QASM Editor/Composer to portray a quantum circuit for an ideal quantum processor

Compile

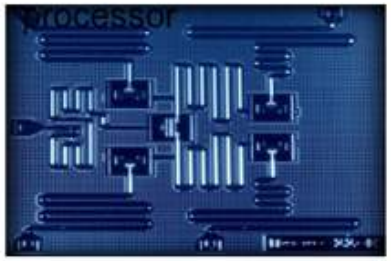
Quantum circuit for real hardware

3.



* subject to hardware constraints
* adds additional gates (swap) to satisfy the hardware constraints

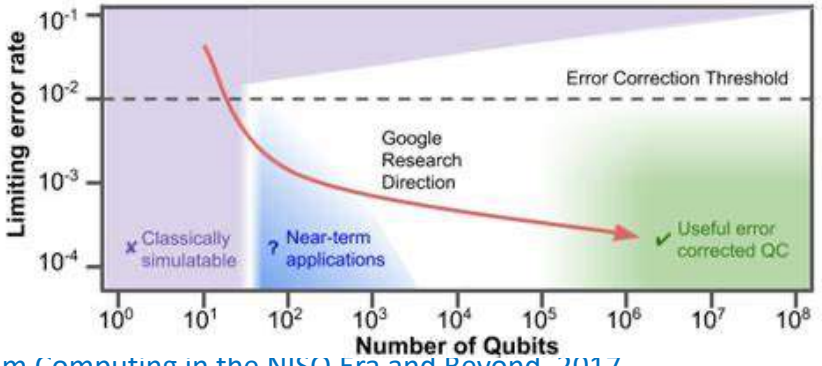
4.



https://www.ibm.com/blogs/research/2018/08/understanding-complexity-quantum-circuit-compilation/

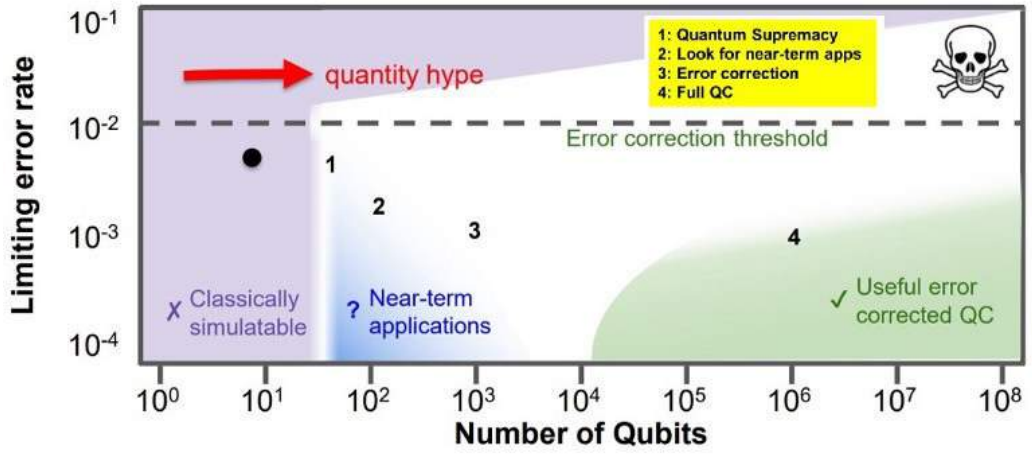
NISQ

- Noisy Intermediate-Scale Quantum[含噪声的中型量子]
  - **Noisy**: devices will be disturbed by what is happening in their environment
    - E.g., small changes in temperature, or stray electric or magnetic fields, can cause the quantum information in the computer to be degraded — a process known as **decoherence**
    - We need to be able to perform error correction — essentially looking at the system to determine which disturbances have occurred, then reversing them
  - **Intermediate-scale**: the number of qubits will most likely be limited to a few hundred, or perhaps a few thousand
    - Long-term: much larger devices, with several millions of qubits and error correction



NISQ (cont.)

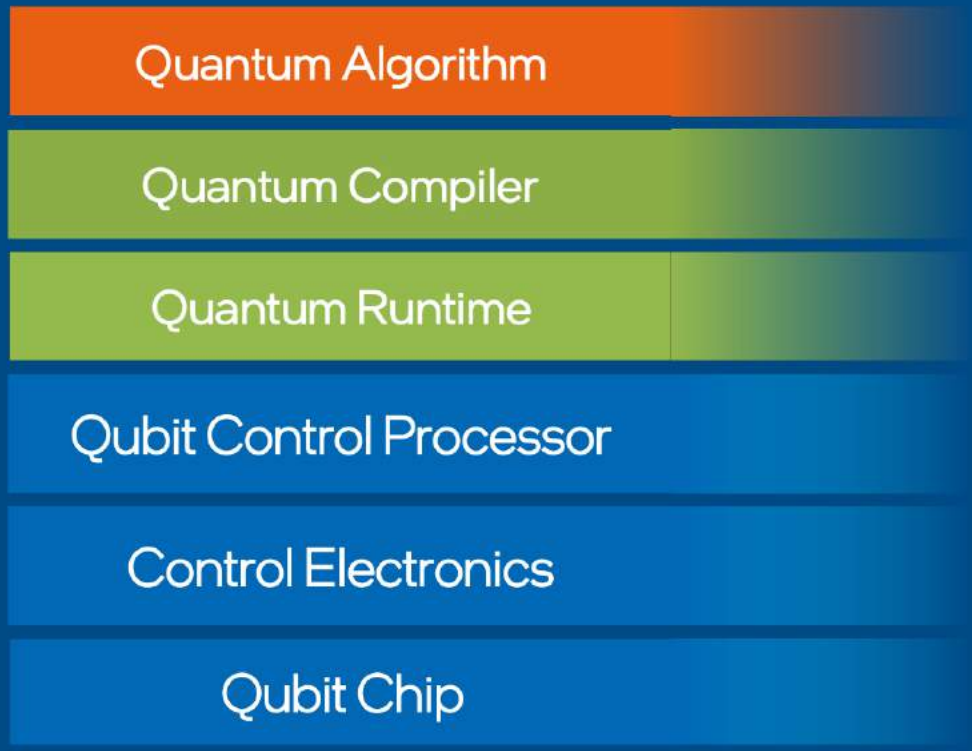
- Quantum computers with 50-100 qubits may be able to perform tasks which surpass the capabilities of today's classical digital computers, , but noise in quantum gates will limit the size of quantum circuits that can be executed reliably
- Quantum technologists should continue to strive for more accurate quantum gates and, eventually, fully fault-tolerant quantum computing



https://www.bgp4.com/2018/10/17/noisy-intermediate-scale-quantum-nisq-technology/

Quantum Computing System

Architecture: Completely New Kind of Compute



Key system challenges for Quantum Practicality

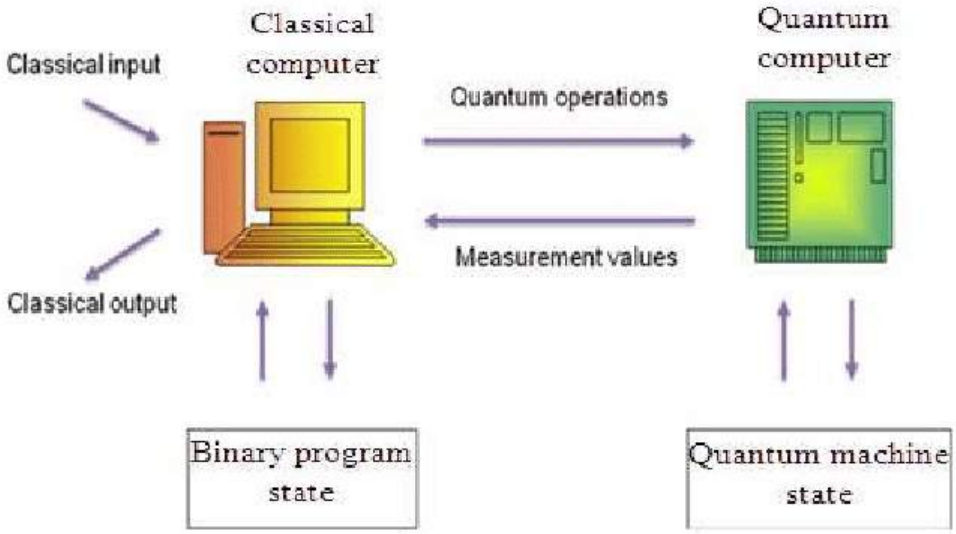
- New execution model
- Error mitigation & resilience
- Scalability
- Interconnect complexity
- Qubit device design

The Future Begins Here

intel labs

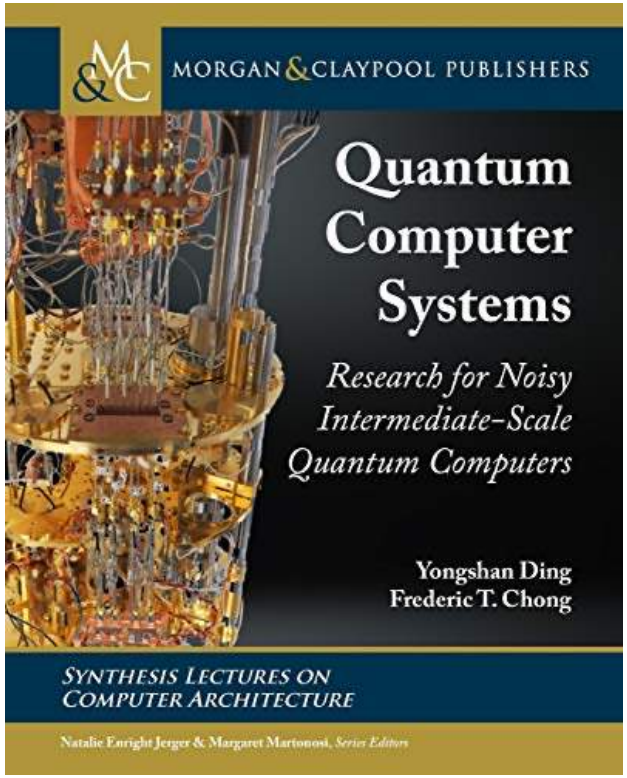
Hybrid Architecture

- Hybrid model to leverage both quantum and classical computation
  - Even as quantum machines scale, quantum algorithms are likely to be specialized, making the quantum device a very **domain-specific accelerator**
  - Most practical applications will still require a combination of general classical and specialized quantum processing to be useful



https://www.sigarch.org/hybrid-quantum-classical-computing/

Researches on Quantum Computer



Quantum Computer Systems: Research for Noisy Intermediate-Scale Quantum Computers

Yongshan Ding and Frederic T. Chong

Synthesis Lectures on Computer Architecture, June 2020, Vol. 15, No. 2 , Pages 1-227 (https://doi.org/10.2200/S01014ED1V01Y202005CAC051)

Yongshan Ding
University of Chicago

Frederic T. Chong
University of Chicago

Abstract

This book targets computer scientists and engineers who are familiar with concepts in classical computer systems but are curious to learn the general architecture of quantum computing systems. It gives a concise presentation of this new paradigm of computing from a computer systems' point of view without assuming any background in quantum mechanics. As such, it is divided into two parts. The first part of the book provides a gentle overview on the fundamental principles of the quantum theory and their implications for computing. The second part is devoted to state-of-the-art research in designing practical quantum programs, building a scalable software systems stack, and controlling quantum hardware components. Most chapters end with a summary and an outlook for future directions. This book celebrates the remarkable progress that scientists across disciplines have made in the past decades and reveals what roles computer scientists and engineers can play to enable practical-scale quantum computing.

Table of Contents: Preface / Acknowledgments / List of Notations / Introduction / Think Quantumly About Computing / Quantum Application Design / Optimizing Quantum Systems--An Overview / Quantum Programming Languages / Circuit Synthesis and Compilation / Microarchitecture and Pulse Compilation / Noise Mitigation and Error Correction / Classical Simulation of Quantum Computation / Concluding Remarks / Bibliography / Authors' Biographies

Researches (cont.)

- HPCA'2021
  - *TILT: Achieving Higher Fidelity on a Trapped-Ion Linear-Tape Quantum Computing Architecture*
  - *QuCloud: A New Qubit Mapping Mechanism for Multi-programming Quantum Computing in Cloud Environment*
  - *Systematic Approaches for Precise and Approximate Quantum State Runtime Assertion*
  - *Faster Schrödinger-style Simulation of Quantum Circuits*
- ASPLOS'2021
  - *Time-Optimal Qubit Mapping*
  - *Orchestrated Trios: Compiling for Efficient Communication in Quantum Programs with 3-Qubit Gates*
  - *Qraft: Reverse Your Quantum Circuit and Know the Correct Program Output*
  - *Logical Abstractions for Noisy Variational Quantum Algorithm Simulation*
  - *CutQC: Using Small Quantum Computers for Large Quantum Circuit Evaluations*

Researches (cont.)

- ISCA'2021
  - *Exploiting Long Distance Interactions and Tolerating Atom Loss in Neutral Atom Quantum Architectures*
  - *Software-Hardware Co-Optimization for Computational Chemistry on Superconducting Quantum Processors*
  - *Designing Calibration and Expressivity-Efficient Instruction Sets for Quantum Computing*
- MICRO'2021
  - *Exploiting Different Levels of Parallelism in the Quantum Control Microarchitecture for Superconducting Qubits*
  - *AutoBraid: A Framework for Enabling Efficient Surface Communication in Quantum Computing*
  - *JigSaw: Boosting Fidelity of NISQ Programs via Measurement Subsetting*
  - *ADAPT: Mitigating Idling Errors in Qubits via Adaptive Dynamical Decoupling*

中山大學 SUN YAT-SEN UNIVERSITY

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

Computer Architecture

计 算 机 体 系 结 构

第25讲：Summary

张献伟

xianweiz.github.io

DCS3013, 12/28/2022

中山大學 SUN YAT-SEN UNIVERSITY

Grading[课时及考核]

- 课时（3学分，54学时）
  - 1-18周，周三 5/6节（14:20-16:00）
  - 10-18周，周一5/6节（14:20-16:00）
  - 地点：教学大楼 C105
- 考核
  - 课堂参与（20%） - 点名、提问、测试
  - 平时作业（40%）- 课下
    - 习题、实践、paper review。。。
  - 课程项目 （40%）
    - simulator
- 课堂
  - 随机点名
    - 缺席优先
  - 随机提问
    - 后排优先
  - 随机测试
    - 不定时间
- 实验/作业
  - 个人完成
    - 杜绝抄袭
  - 按时提交
    - 超算习堂

Covered Contents

Week/Date	Topic
wk1: Aug 31	Overview
wk2: Sep 7	Quantitative Approach (1)
wk3: Sep 14	Quantitative Approach (2), ISA & ILP (1)
wk4: Sep 21	ISA & ILP (2)
wk5: Sep 28	ISA & ILP (3)
wk6: Oct 5	NO CLASS
wk7: Oct 12	simulator, gem5
wk8: Oct 19	ISA & ILP (4)
wk9: Oct 26	ISA & ILP (5), Data Level Parallelism (1)[online]
wk10: Oct 31	Data Level Parallelism (2)[online]
wk10: Nov 2	Data Level Parallelism (3)[online]
wk11: Nov 7	Data Level Parallelism (4)[online]
wk11: Nov 9	Data-level Parallelism (5), Memory (1)[online]
wk12: Nov 14	Memory (2)[online]

Week/Date	Topic
wk12: Nov 16	Memory (3)[online]
wk13: Nov 21	Memory (4)[online]
wk13: Nov 23	Memory (5), Thread-level Parallelism (1)[online]
wk14: Nov 28	Thread-level Parallelism (2)[online]
wk14: Nov 30	Thread-level Parallelism (3)[online]
wk15: Dec 5	Thread-level Parallelism (4)[online]
wk15: Dec 7	Thread-level Parallelism (5)[online]
wk16: Dec 12	Thread-level Parallelism (6), WSC & Interconnect (1)[online]
wk16: Dec 14	WSC & Interconnect (2)[online]
wk17: Dec 19	WSC & Interconnect (3), Domain-specific Arch (1)[online]
wk17: Dec 21	Domain-specific Arch (2)[online]
wk18: Dec 26	Domain-specific Arch (3)
wk18: Dec 28	Summary & Advanced

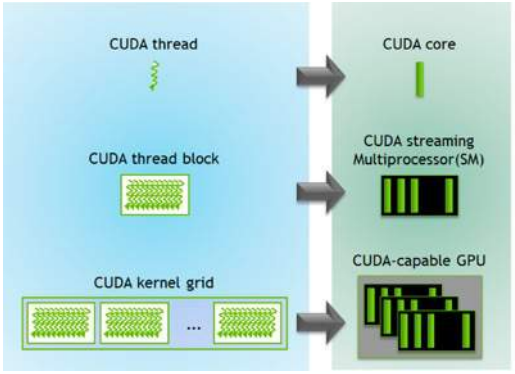
I: ISA & ILP

- ILP Challenge: overcoming data and control dependencies

Technique	Reduces	Section
Forwarding and bypassing	Potential data hazard stalls	C.2
Simple branch scheduling and prediction	Control hazard stalls	C.2
Basic compiler pipeline scheduling	Data hazard stalls	C.2, 3.2
Basic dynamic scheduling (scoreboarding)	Data hazard stalls from true dependences	C.7
Loop unrolling	Control hazard stalls	3.2
Advanced branch prediction	Control stalls	3.3
Dynamic scheduling with renaming	Stalls from data hazards, output dependences, and antidependences	3.4
Hardware speculation	Data hazard and control hazard stalls	3.6
Dynamic memory disambiguation	Data hazard stalls with memory	3.6
Issuing multiple instructions per cycle	Ideal CPI	3.7, 3.8
Compiler dependence analysis, software pipelining, trace scheduling	Ideal CPI, data hazard stalls	H.2, H.3
Hardware support for compiler speculation	Ideal CPI, data hazard stalls, branch hazard stalls	H.4, H.5

II: DLP & GPU

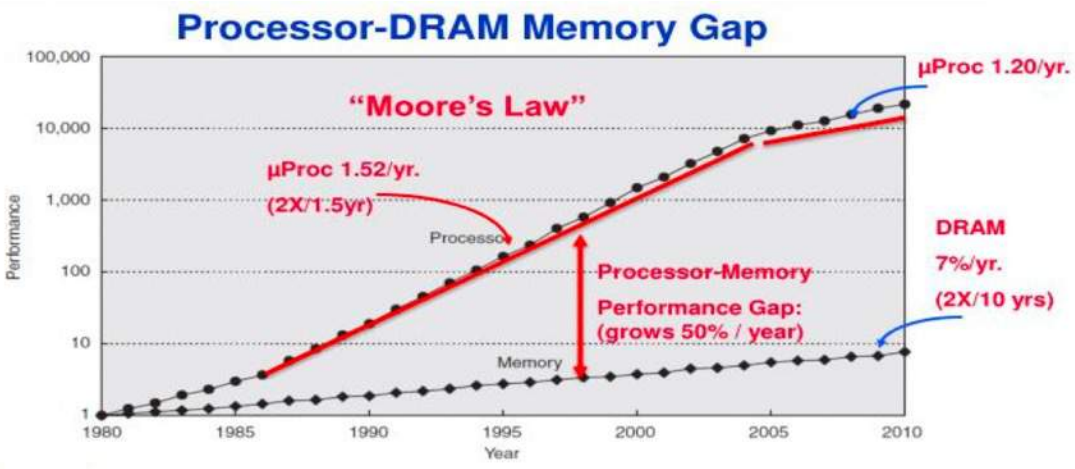
- Data level parallelism
  - SIMD: operates on multiple data with on single instruction
    - AVX-512 on Intel CPU is the typical example
  - SIMT: consists of multiple scalar threads executing in a SIMD manner
    - GPU is the example with threads executing the same instruction
- GPU hardware and thread organization
  - Device → SM → SIMD/Partition → Core
  - Grid → Block → Warp → Thread



- GPU programming
  - Streams to support concurrency
  - Memory hierarchy and usage (thread, cache/smem, global)
  - Advanced topics: virtualization, divergence, nvlink, etc

III: Memory

- Memory wall issue
  - On modern machines, most programs that access a lot of data are memory bound
- Cache
  - Temporal and spatial locality
  - Organization, management and advanced optimizations
- DRAM and NVM
  - Structure, variants, scaling issues and emerging memories

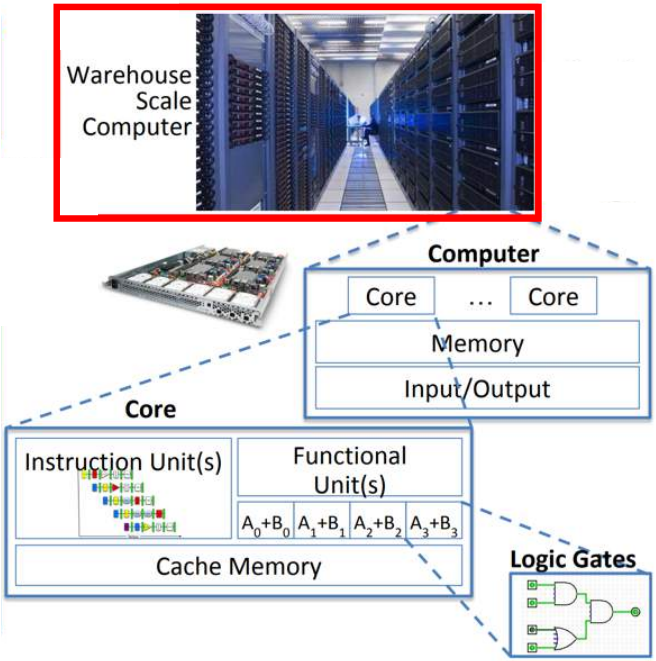


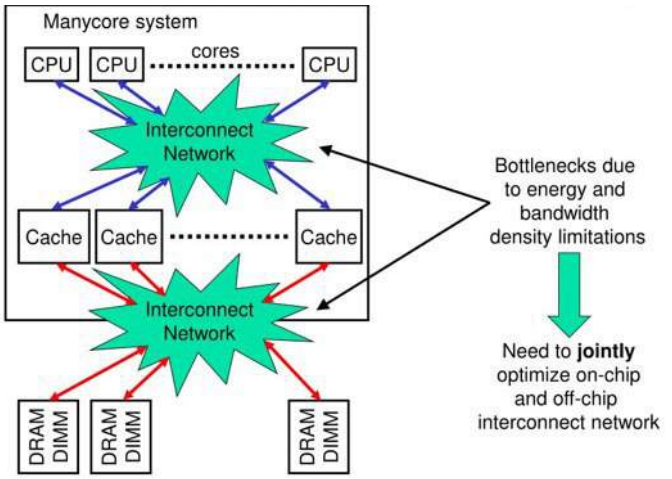
IV: TLP

- Multiprocessors with thread-level parallelism
  - Sharing memory, having private caches
- Cache coherence
  - **Snooping**: every cache block is accompanied by the sharing status of that block
    - All cache controllers monitor the shared bus so they can update the sharing status of the block, if necessary
  - **Directory-based**: a single location (directory) keeps track of the sharing status of a block of memory
    - Reduce storage and communication overheads
- Memory consistency
  - **Sequential consistency**: maintains all four memory operation orderings (W→R, R→R, R→W, W→W)
  - **Relaxed consistency**: allows certain orderings to be violated
    - TSO, PSO, RC

V: WSC & Interconnect

- Warehouse scale computer
  - Request-level parallelism
  - Server -> rack -> array -> WSC
  - Power usage effectiveness (PUE)
- Interconnection network
  - System area and on-chip
  - Topologies: Bus, Xbar, Multistage, Star, Linear, Mesh, Tree
  - Compute Express Link (CXL)
    - Unified, coherent memory space
    - High speed, low latency





VI: DSA

- DSA will co-exist with general-purpose architecture
  - Heterogeneous: CPU + GPU + accelerator
- DSA design guidelines
  - Dedicated memories, larger ALU, easy parallelism, smaller data size, domain-specific language
- Google TPU
  - DNNs, matrix unit, on-chip memory, systolic execution, ISA
  - Roofline performance model
    - Tie architecture (peak computation and bandwidth) to application (arithmetic intensity)
- Quantum computing

Goals[课程目标]

- This course covers HW/SW, and the interface[内容覆盖]
  - We will focus on performance analysis and design tradeoffs
- Two key goals of this course are[主要目标]
  - To understand how hardware components works with the software layer and how decisions made in hardware affect the software/programmer
  - To enable you to be comfortable in making design and optimization decisions that cross the boundaries of different layers and system components

- Two other goals of this course[额外目标]
  - Enable you to think critically
  - Enable you to think broadly

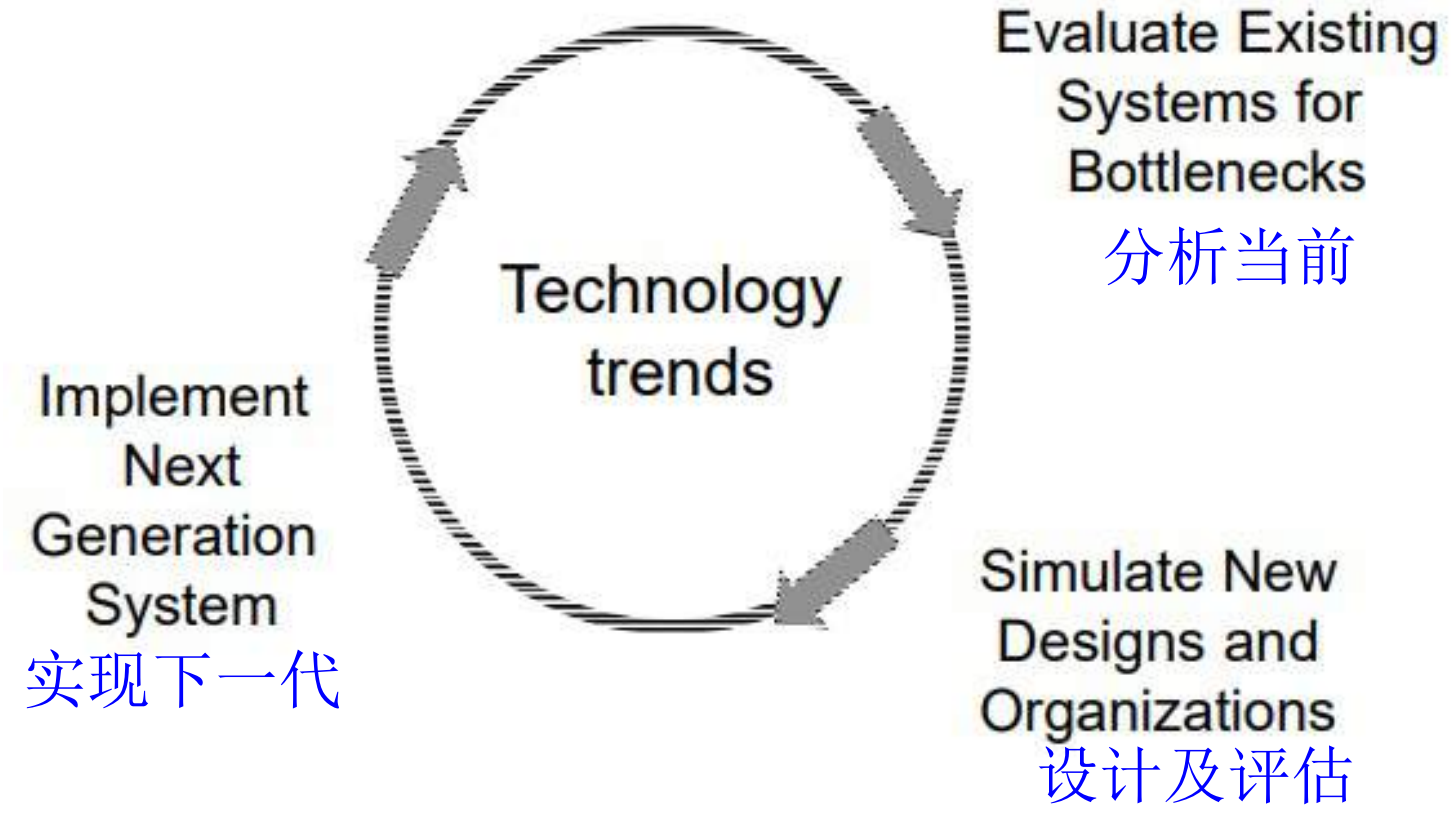
Design Goals[设计目标]

- 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

Methodology: Design/Evaluation[方法]

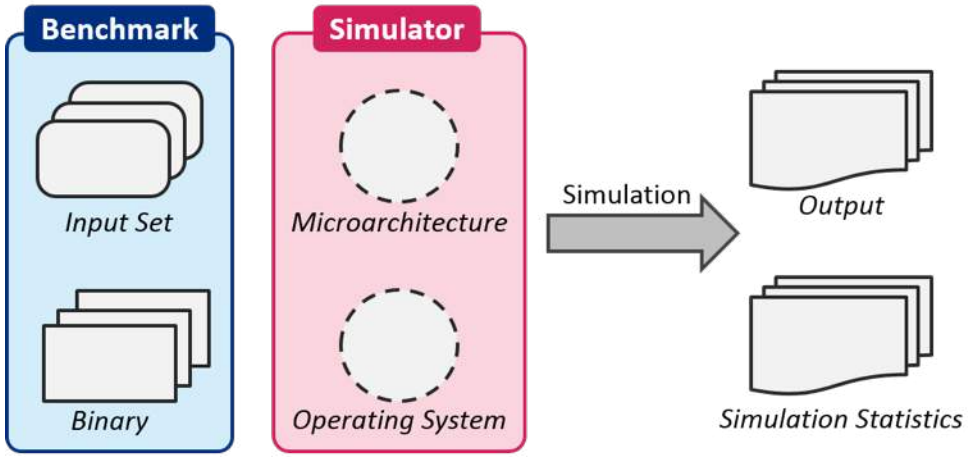


Quantitative Principles (§1.8)[量化原则]

- 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

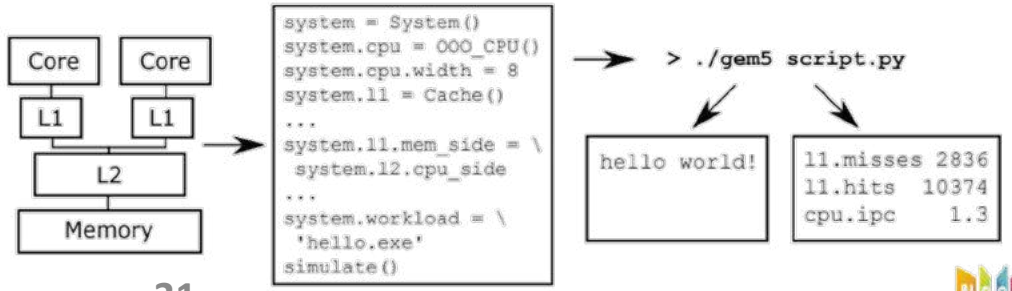
Simulator[模拟器]

- 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



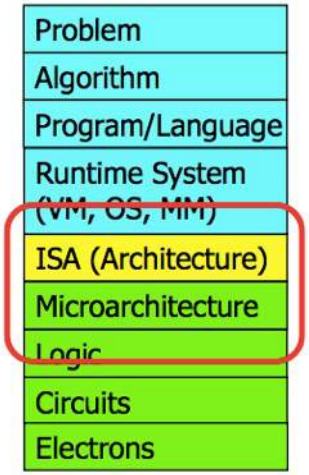
Example Simulator: gem5

- 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)

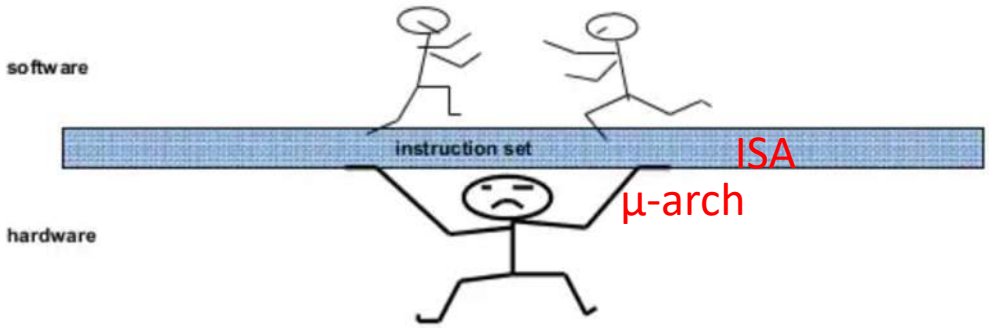


https://pages.cs.wisc.edu/~sinclair/courses/cs752/fall2020/handouts/lecture/archSim.pdf

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



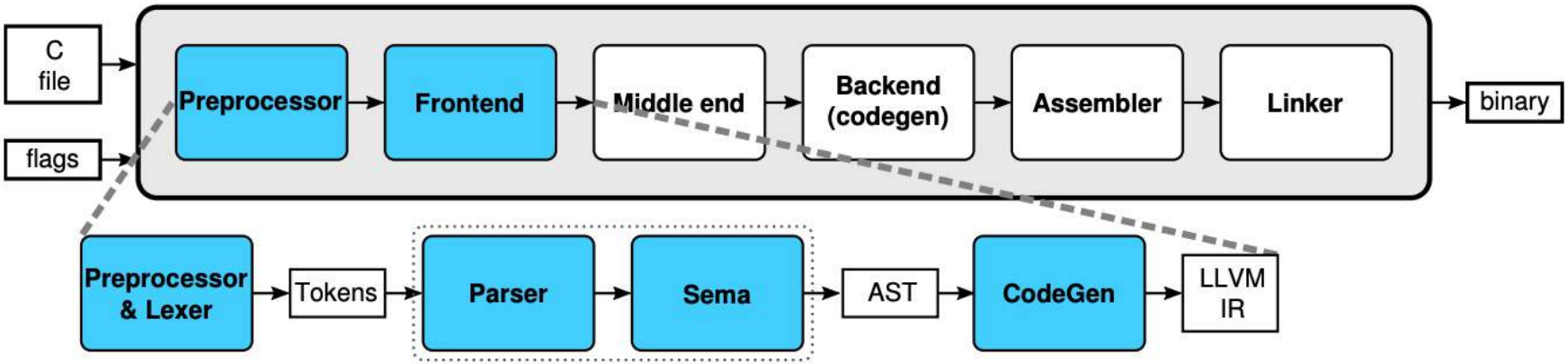
https://image.slideserve.com/466455/instruction-set-design-l.jpg

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

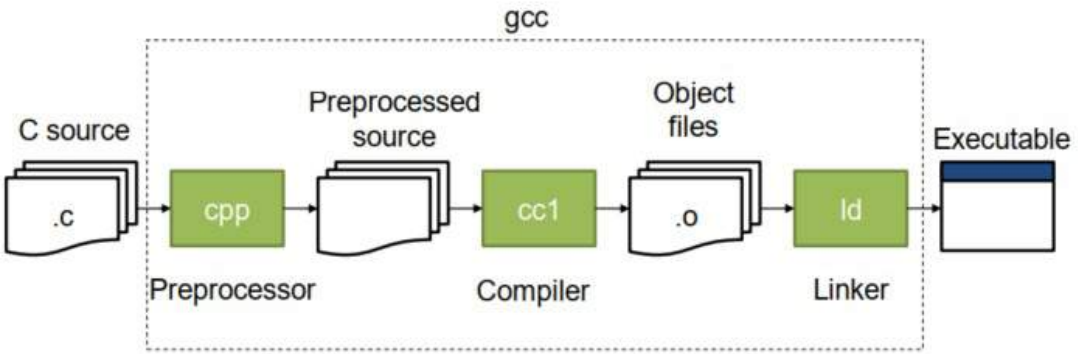
Compiler




```c
#include <stdio.h>

int main()
{
   printf("Hello World!\n");

   return 0;
}
```




```
55
48 89 e5
bf d0 05 40 00
e8 d5 fe ff ff
b8 00 00 00 00
5d
c3
```

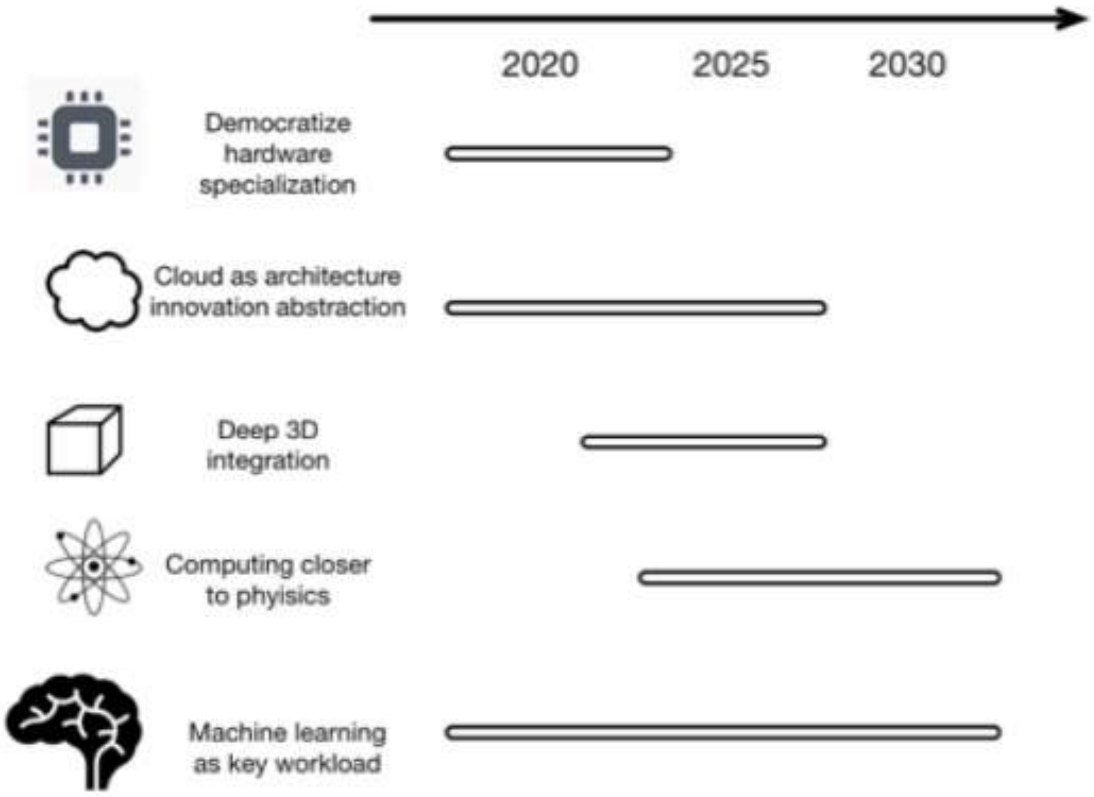
gcc –E hello.c -o hello.i gcc -S hello.i -o hello.s gcc hello.o –o hello

Golden Age for CA

- 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]**

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)