ECE 5997
Hardware Accelerator Design & Automation
Fall 2021

Introduction

Zhiru Zhang
School of Electrical and Computer Engineering

Cornell University

CSL

Agenda

- Important logistics
- Course motivation
- Basics of algorithm analysis and graphs

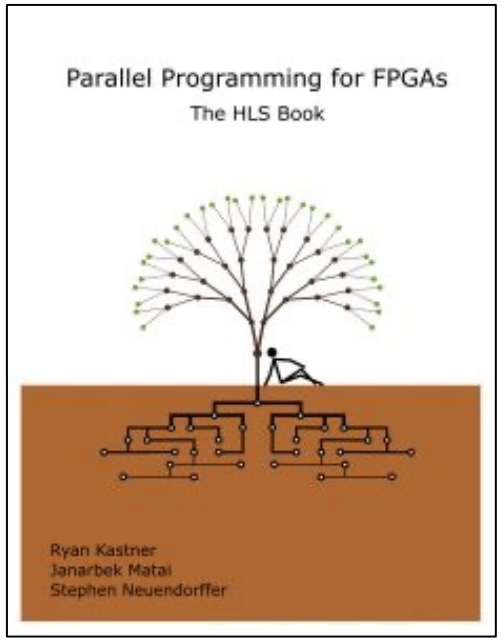
Class Resources

- Course website
  - https://www.csl.cornell.edu/~zhiruz/5997
  - Lectures slides, handouts, and other readings

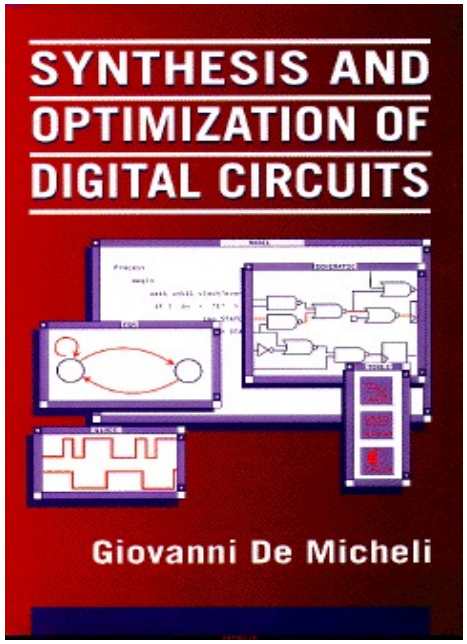
- Ed Discussion
  - Announcements and Q&A
  - Enrollment information to come

- CMS: course management system
  - Assignments and grades
  - Electronic submissions required

Course Texts



[e-book available online]



Get 1st edition
[Overhead slides available online]


Graph-Based Algorithms for Boolean Function Manipulation

Selected paper & software manuals

Seeking Help After Class

- Ed Discussion
  - Questions on lectures, assignments, projects, etc.
  - Monitored by course staff
    - PhD TAs: Shaojie Xiang (sx233), Yichi Zhang (yz2499)
- Instructor email
  - Personal issues/appointment
- (TA-led) office hours
  - Tuesday 4:40-5:55pm, online (same zoom link)

Course Organization

- Refer to syllabus for course organization details

1. Course Information

Lectures: Friday 04:40-05:55pm
OH/Tutorial: Tuesday 04:40-5:55pm
Instructor: Zhiru Zhang, zhiruz@cornell.edu
Credits: 1 Credit

Course Texts:
○ R. Kastner, J. Matai, and S. Neuendorffer, Parallel Programming for FPGAs, arXiv, 2018.

2. Course Description

Targeted specialization of functionality in hardware has become arguably the best means to achieving improved compute performance and energy efficiency for a plethora of emerging applications. Unfortunately, it is a very unproductive practice to design and implement special-purpose accelerators using the conventional register transfer level (RTL) methodology. For this reason, both academia and industry are seeing an increasing use of high-level synthesis (HLS) to automatically generate hardware accelerators from software programs.

The course provides an introduction to the hardware accelerator design principles and the modern HLS design methodologies and tools, focusing on FPGA targets. Specific topics include C-based HLS design methods, hardware specialization, scheduling, pipelining, resource sharing, and case studies on deep learning acceleration. Commercial C-to-FPGA tools will be provided to the students to implement real-life image/video processing and machine learning applications on programmable system-on-chips that tightly integrate CPU and FPGA devices.

2.1. Prerequisites

This course assumes the student has a working knowledge of C/C++ and familiarity with basic concepts of digital logic and computer architecture, such as sequential circuits, timing analysis, pipelining, etc. A knowledge of basic algorithms and data structures is preferred. Experiences with RTL design for either ASICs or FPGAs would be helpful, although not required.

2.2. Target Audience and Learning Outcomes

This course is open to graduate students and senior undergraduates, who are interested in (1) learning application-/domain-specific hardware acceleration and (2) understanding the capabilities of current HLS tools and design methodologies. Upon completion of this course, students will be able to use HLS tools to design realistic hardware accelerators on FPGAs.

3. Course Organization

This course includes a combination of lectures (Friday) and a few TA-led tutorials (Tuesday) that cover the following topics.

This Course is About Hardware/Software Co-Design

- Specifying algorithms in software (SW) programming languages
- Compiling SW descriptions into special-purpose hardware (HW) architectures
  - Automatic compilation & synthesis techniques
  - Performance, area, power trade-offs
- Realizing SW & HW on reconfigurable system-on-chips
  - FPGA-targeted implementation

This Course Introduces EDA

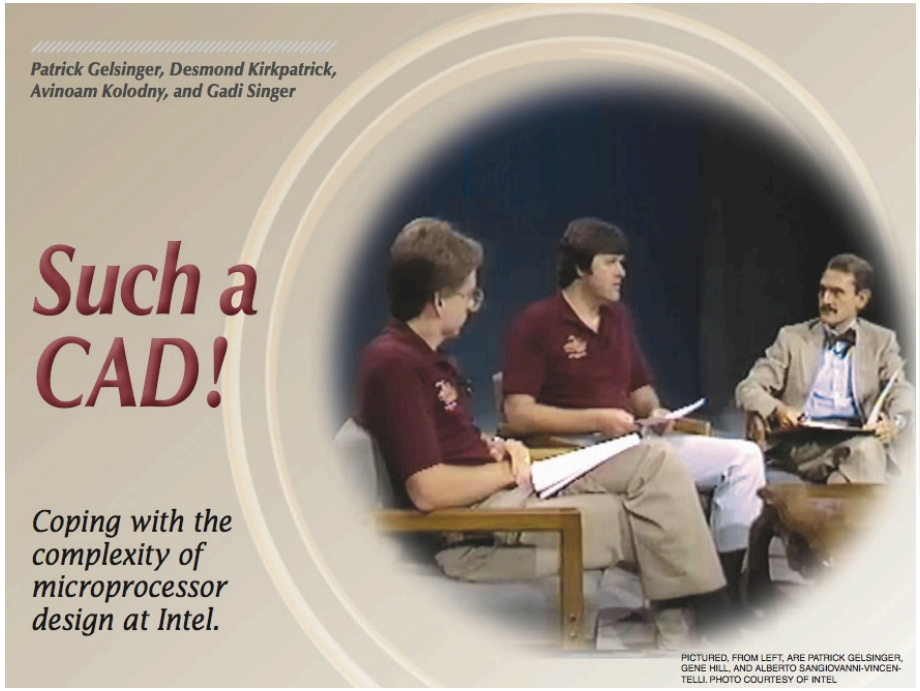
Electronic Design Automation

- A general methodology for refining **a high-level description down to a detailed physical implementation** for designs ranging from
  - integrated circuits (including system-on-chips),
  - printed circuit boards (PCBs),
  - and electronic systems

- **Modeling**, **synthesis**, and **verification** at every level of **abstraction**

[source: NSF'09 EDA Workshop]

High Impact of EDA



[source] Patrick Gelsinger, Desmond Kirkpatrick, Avinoam Kolodny, and Gadi Singer. "Such a CAD!" *IEEE Solid-State Circuits Magazine*, 2010.

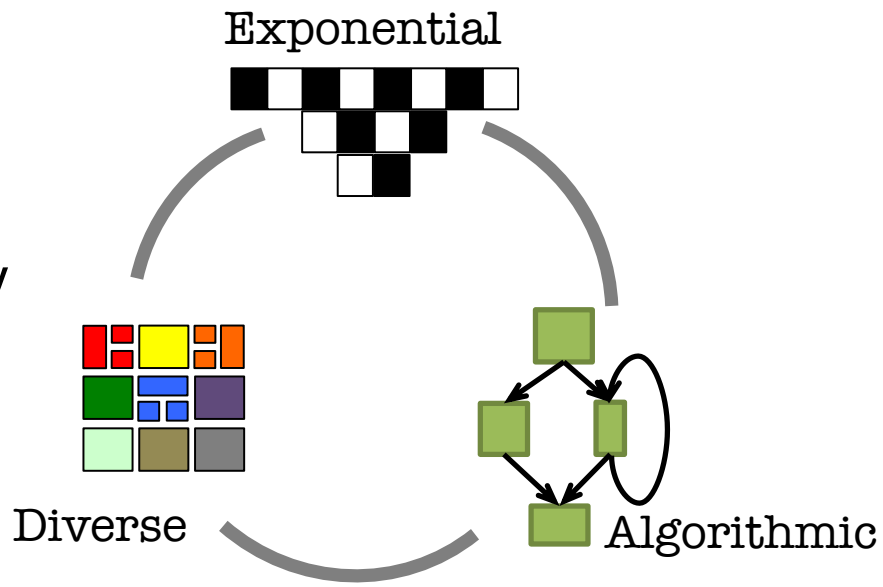
TABLE 1. INTEL PROCESSORS, 1971–1993.

PROCESSOR	INTRO DATE	PROCESS	TRANSISTORS	FREQUENCY
4004	1971	10 μm	2,300	108 KHz
8080	1974	6 μm	6,000	2 MHz
8086	1978	3 μm	29,000	10 MHz
80286	1982	1.5 μm	134,000	12 MHz
80386	1985	1.5 μm	275,000	16 MHz
Intel 486 DX	1989	1 μm	1.2 M	33 MHz
Pentium	1993	0.8 μm	3.1 M	60 MHz

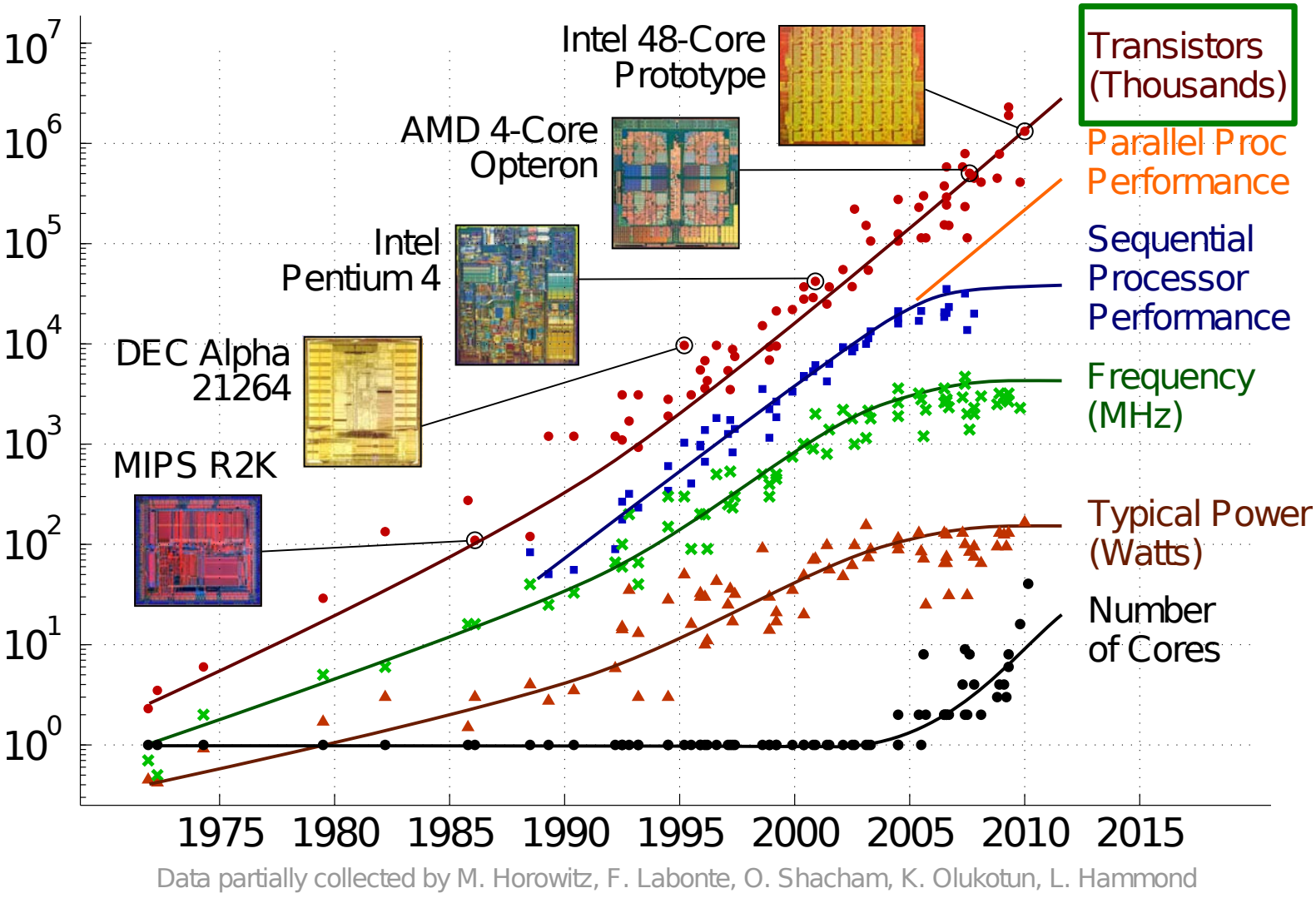
"This incredible growth rate could not be achieved by hiring an exponentially growing number of design engineers. It was fulfilled by adopting new design methodologies and by introducing innovative design automation software at every processor generation."

E-D-A: My Other Interpretation

- **E**xponential
  - in complexity (or **E**xtreme scale)
- **D**iverse
  - increasing system heterogeneity
  - multi-disciplinary
- **A**lgorithmic
  - intrinsically computational

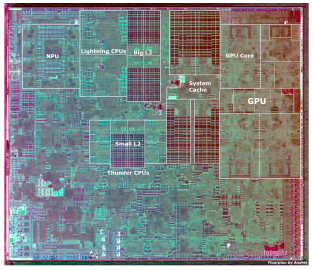


Exponential: Moore's Law

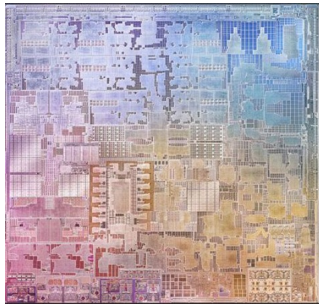


[Figure credit: Christopher Batten, Cornell]

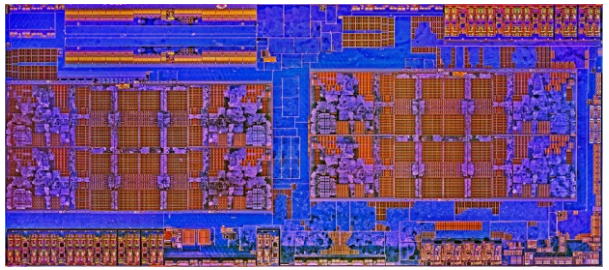
Era of Billion-Transistor Chips



Apple A13
~8B transistors

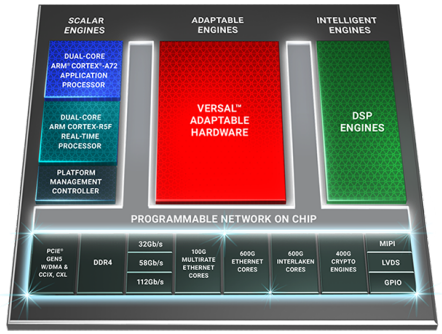


Apple M1
~16B transistors



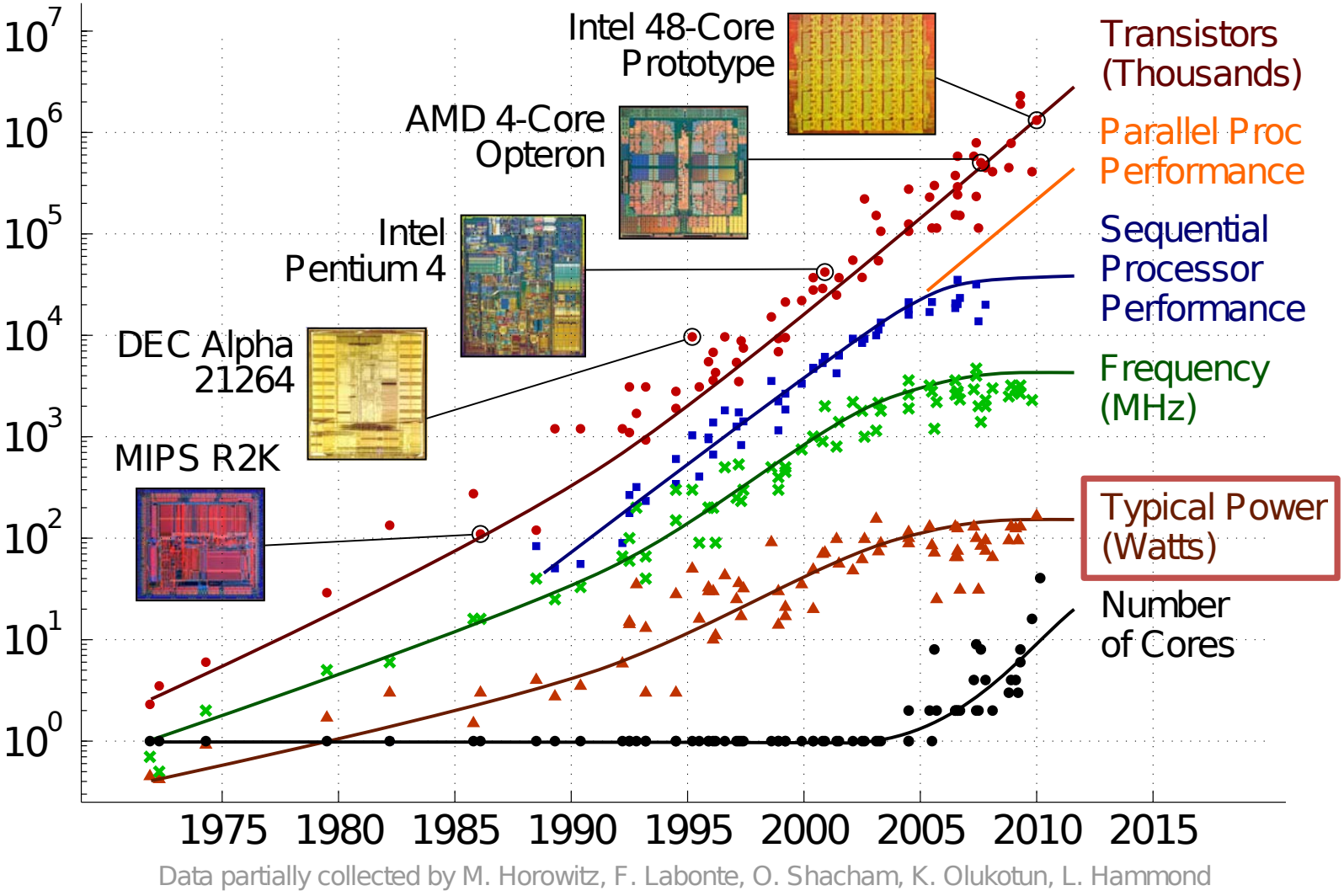
AMD EPYC Rome
~39B transistors

NVIDIA A100 Ampere
~54B transistors



Xilinx Versal VP1802
~92B transistors

End of Dennard Scaling:
Power Becomes the Limiting Factor



[Figure credit: Christopher Batten, Cornell]

Power-Constrained Modern Computers

<<1W/chip ~1W/chip ~15W/chip ~50W/chip ~100W/chip >100W/chip

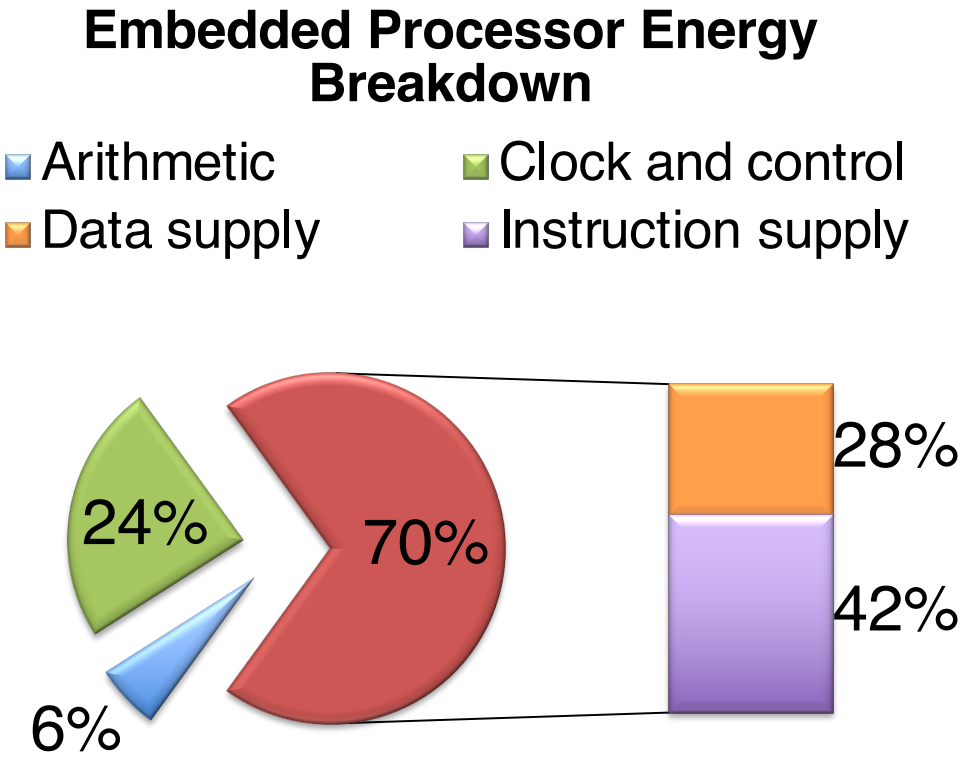
$$Power = \frac{Energy}{Second} = \frac{Energy}{Op} \times \frac{Ops}{Second}$$

- **Energy efficiency (Ops/Joule) must improve!**
- **Limitations of general-purpose multicore scaling**
  - Amdahl's law
  - Dark silicon

Inefficiency of General-Purpose Computing

- Typical energy overhead for every 10pJ arithmetic operations
  - 70pJ on instruction supply
  - 47pJ on data supply

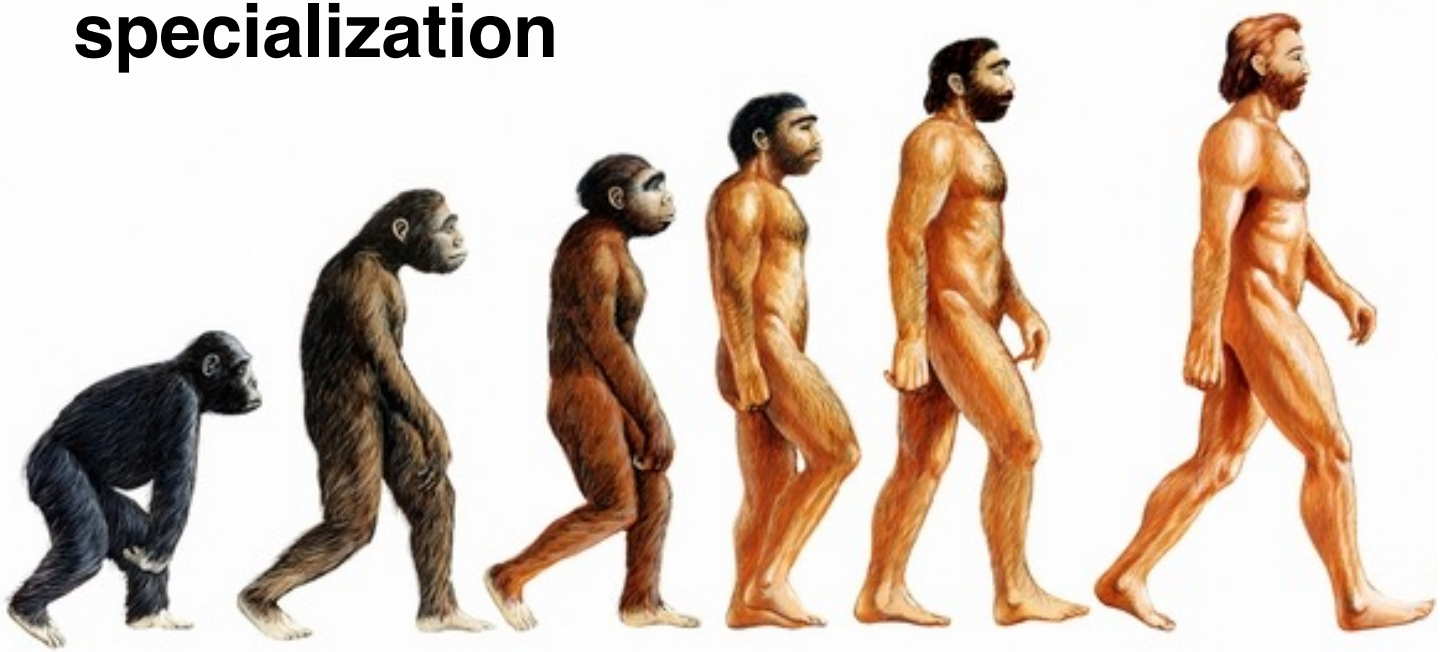
Plus, only 59% of the instructions are arithmetic



[source: Dally et al. Efficient Embedded Computing, IEEE'08]

Advance of Civilization

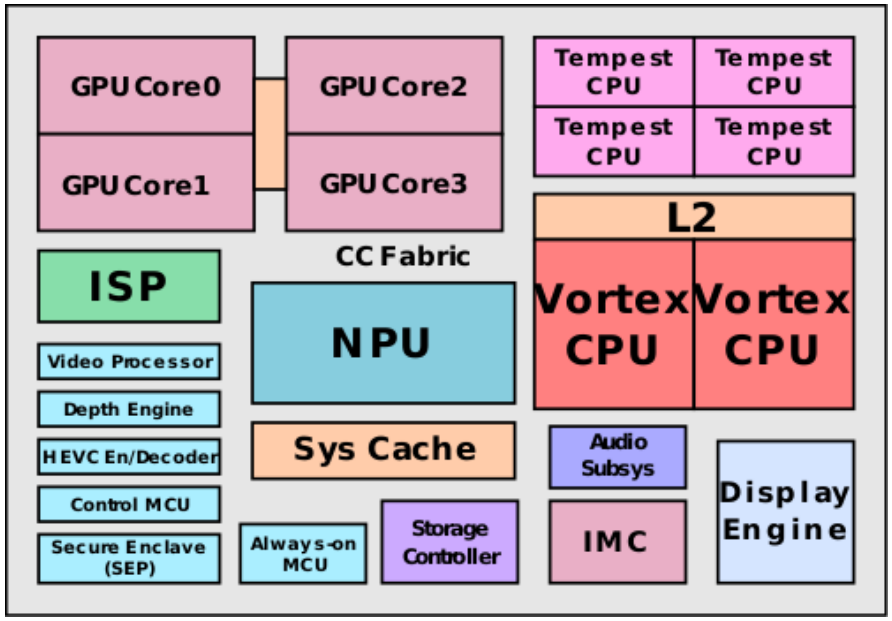
- For humans, Moore's Law scaling of the brain has ended a long time ago
  - Number of neurons and their firing rate did not change significantly
- Remarkable advancement of civilization via **specialization**



?

Computers are Following the Same Path: Diverse Range of Integrated Functionalities

System on chip (SoC)



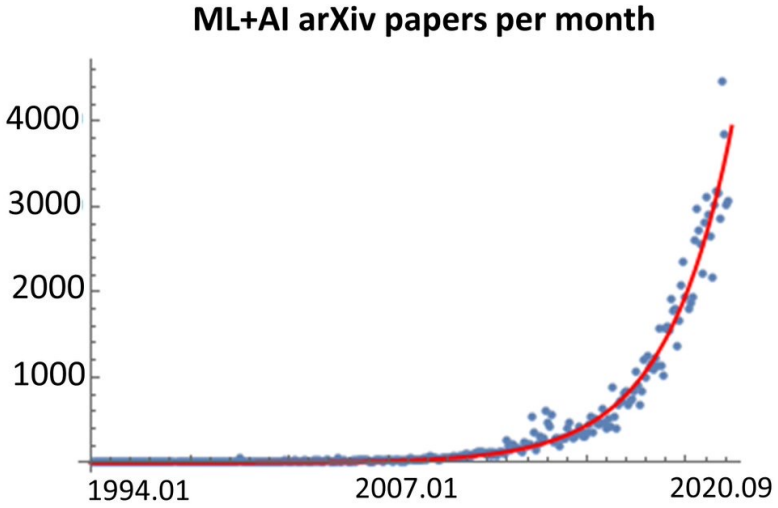
Apple 12 (iPhone X)

- Modern SoCs integrate a rich set of **special-purpose accelerators**
  - Speed up critical tasks
  - Reduce power consumption and cost
  - Increase energy efficiency

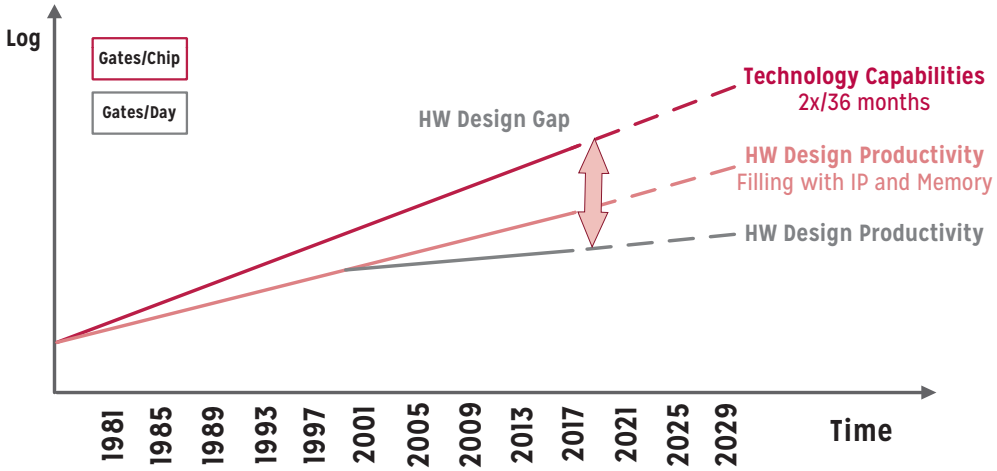
Increasing Specialization Demands Higher Design Productivity

Can custom hardware evolve fast enough to keep up?

- Target of specialization is moving rapidly

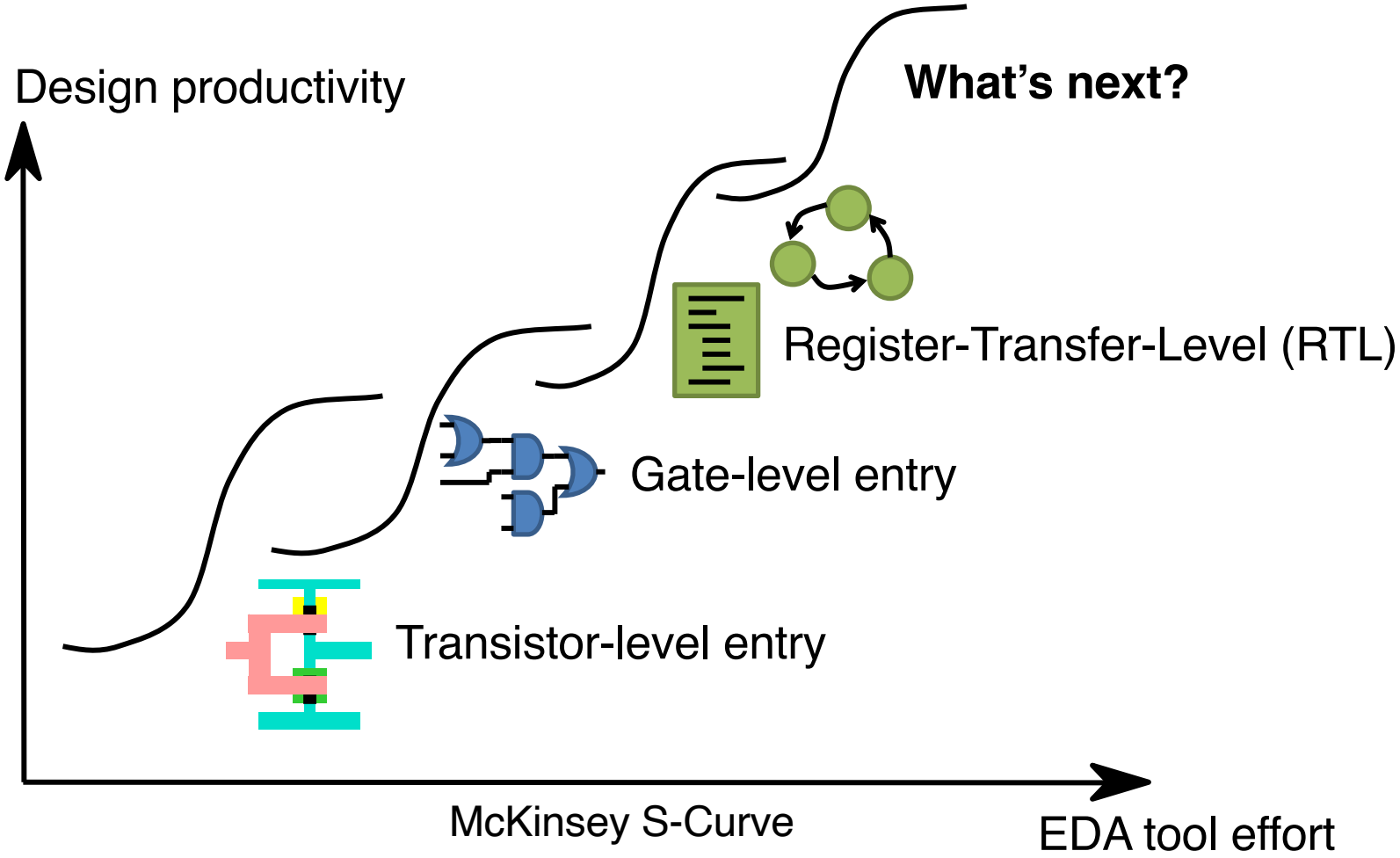


Number of machine learning papers published on arXiv has outpaced Moore's Law
[Dean et al., IEEE Micro 2018]



The design productivity gap
[Source: Workshops on Extreme Scale Design Automation: Challenges and Opportunities for 2025 and Beyond]

Evolution of Design Abstraction



[source: Kurt Keutzer, UCB]

Motivation for High-Level Synthesis (HLS)

```
module dut(rst, clk, q);
  input rst;
  input clk;
  output q;
  reg [7:0] c;

  always @ (posedge clk)
  begin
    if (rst == 1b'1) begin
      c <= 8'b00000000;
  end
  else begin
    c <= c + 1;
  end

  assign q = c;
endmodule
```

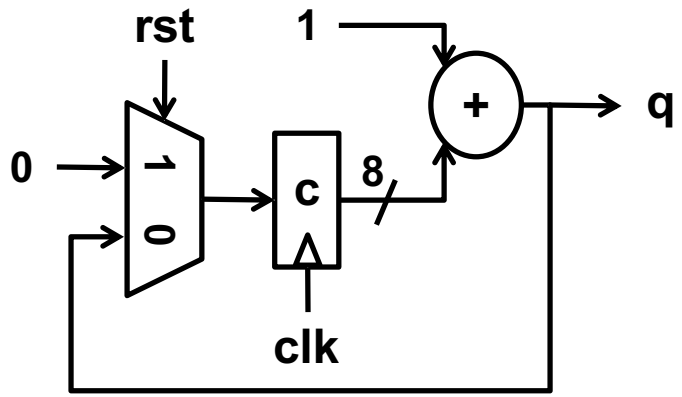
RTL Verilog

vs.

```
uint8 dut() {
  static uint8 c;
  c+=1;
}
```

⇩ **automatically?**

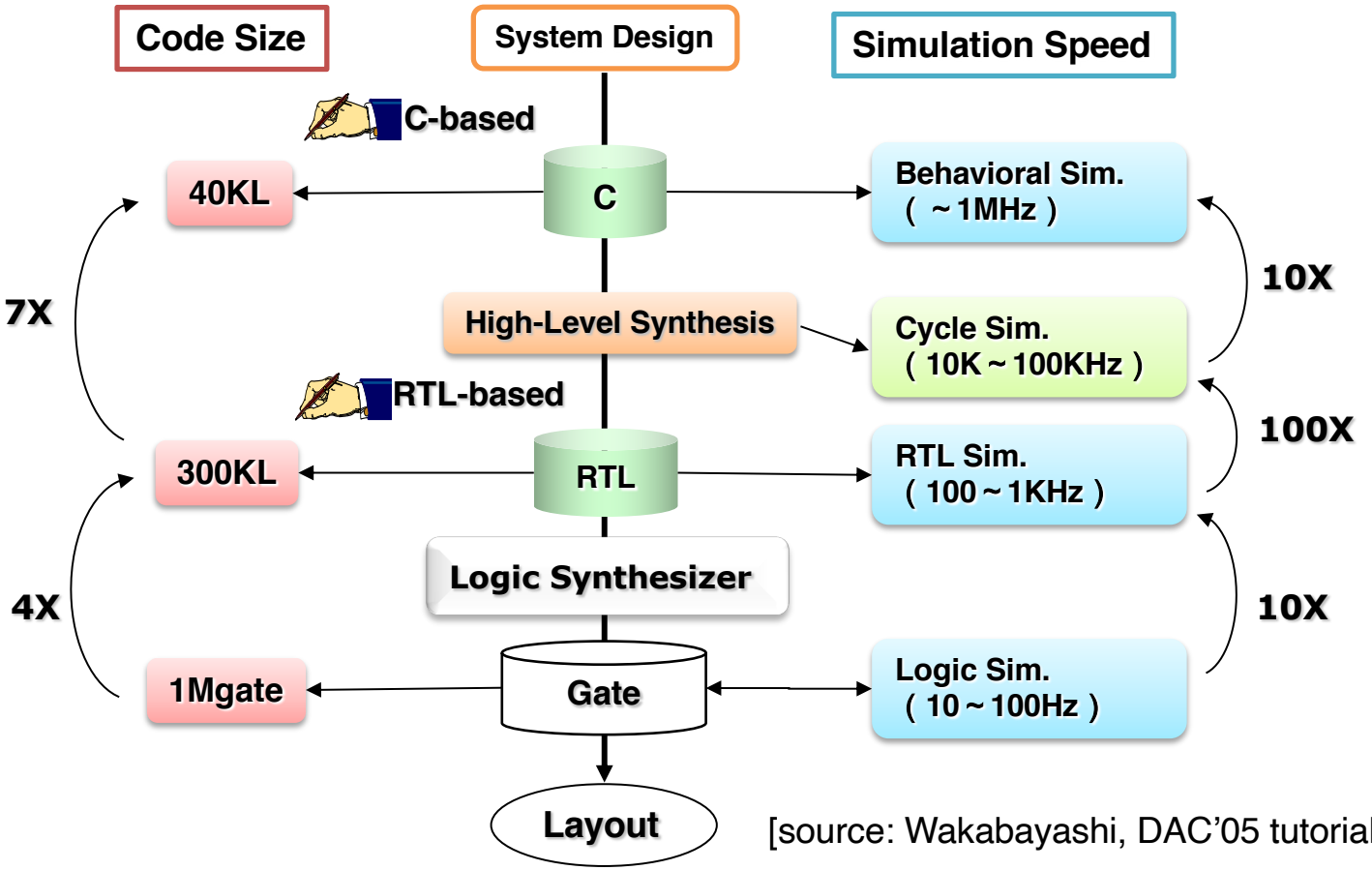
⇨



An 8-bit counter

High-Level Design Automation to Manage Design Complexity

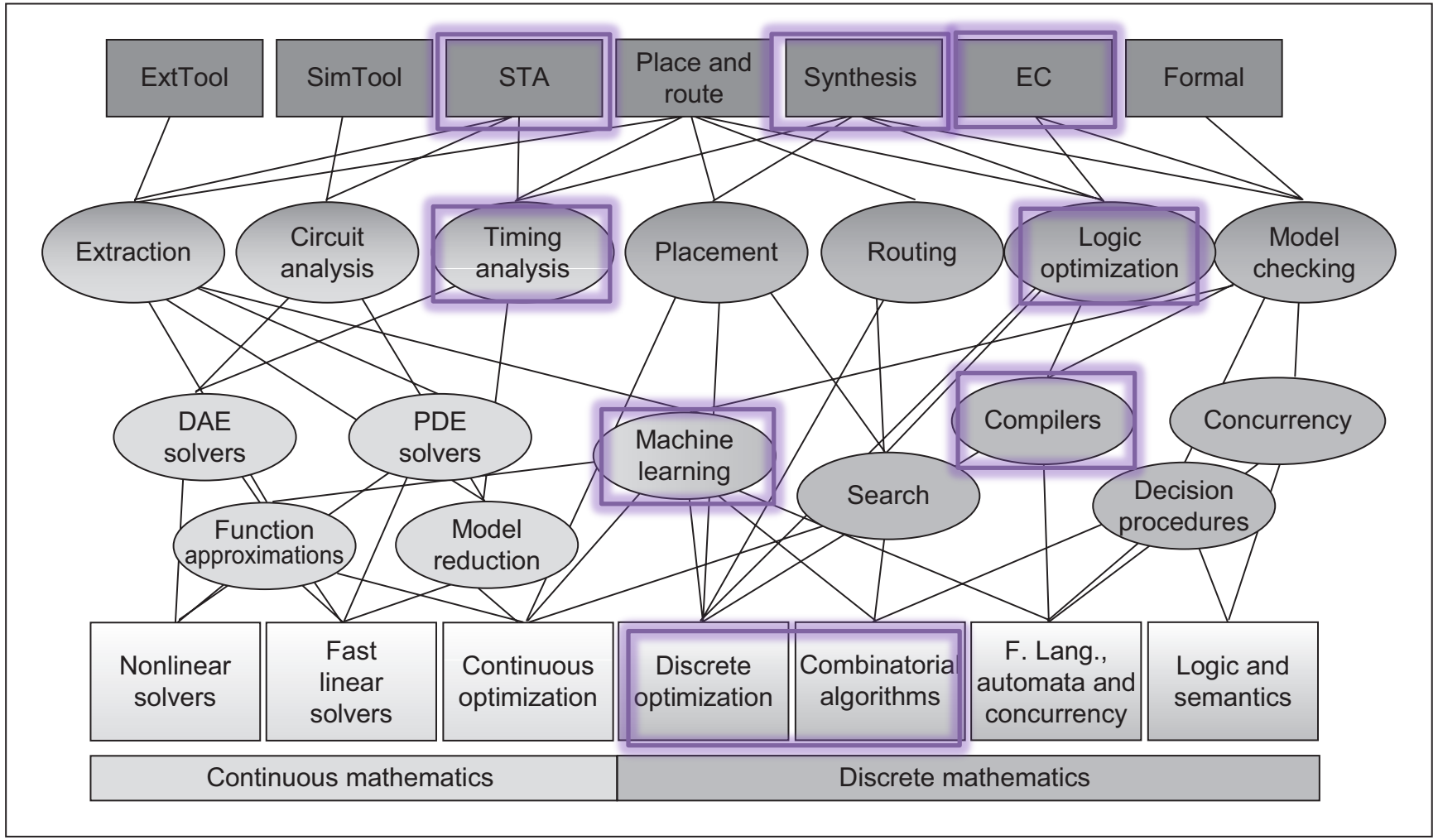
- Significant code size reduction
- Shorter simulation/verification cycle



[source: Wakabayashi, DAC'05 tutorial]

Algorithms Drive Automation

Topics touched in this course



Key Algorithms in EDA

[source: Andreas Kuehlmann, Synopsys Inc.]

Analysis of Algorithms

- Need a systematic way to compare two algorithms
  - Execution time is typically the most common criterion used
  - Space (memory) usage is also important in most cases

  - But difficult to compare in practice since algorithms may be implemented on different machines, use different languages, etc.
  - Additionally, runtime is usually input-dependent

- **big-O** notation is widely used for asymptotic analysis
  - Complexity is represented with respect to some natural & abstract measure of the problem size *n*

Big-O Notation

- Express run time as a function of input size n
  - Run time $F(n)$ is of order $G(n)$, written as $F(n) = \mathbf{O}(G(n))$ when
    - $\exists n_0, \forall n \geq n_0, F(n) \leq KG(n)$ for some constant K
  - F will not grow larger than G by more than a constant factor
  - G is often called an "**upper bound**" for F
- Interested in the worst-case input & the growth rate for large input size

Big-O Notation (cont.)

- How to determine the order of a function?
  - Ignore lower order terms
  - Ignore multiplicative constants

  - Examples:

    $3n^2 + 6n + 2.7$ is $O(n^2)$

    $n^{1.1} + 10000000000n$ is $O(n^{1.1})$, $n^{1.1}$ is also $O(n^2)$

  - **$n! > C^n > n^C > \log n > \log\log n > C$**

    $\Rightarrow$ $n! > n^{10}$; $n \log n > n$; $n > \log n$

- What do asymptotic notations mean in practice?
  - If algorithm A is $O(n^2)$ and algorithm B is $O(n \log n)$, we usually say algorithm B is **more scalable**.

Exponential Growth

- Consider a 1GHz processor (1ns per clock cycle) running 2^n operations (assuming each op requires one cycle)

n	2^n	1ns (/op) x 2^n
10	10^3	1 us
20	10^6	1 ms
30	10^9	1 s
40	10^{12}	16.7 mins
50	10^{15}	11.6 years
60	10^{18}	31.7 years
70	10^{21}	31710 years

NP-Complete

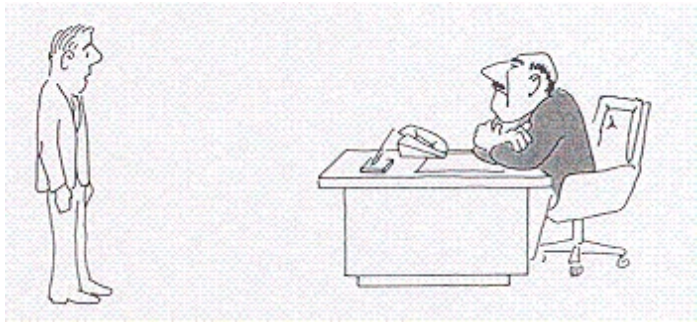
- The class **NP-complete** (NPC) is the set of decision problems which we “believe” there is no polynomial time algorithms (hardest problem in NP)

- **NP-hard** is another class of problems, which are at least as hard as the problems in NPC (also containing NPC)

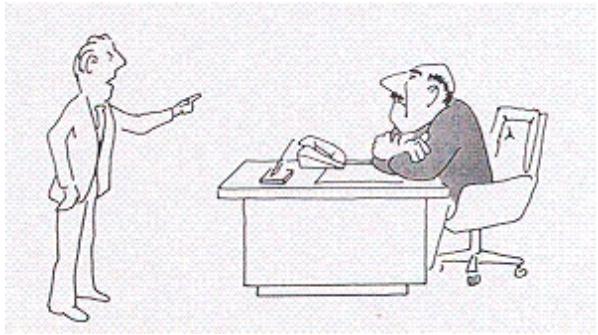
- If we know a problem is in NPC or NP-hard, there is (very) little hope to solve it exactly in an efficient way

How to Identify an NP-Complete Problem

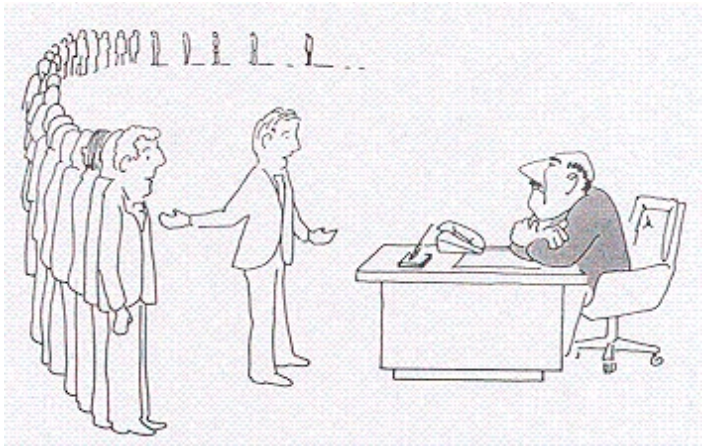
- I can't find an efficient algorithm, I guess I'm just too dumb.



- I can't find an efficient algorithm, because no such algorithm is possible.



- I can't find an efficient algorithm, but neither can all these famous people.



[source: Computers and Intractibility by Garey and Johnson]

Problem Intractability

- Most of the nontrivial EDA problems are intractable (NP-complete or NP-hard)
  - Best-known algorithm complexities that grow exponentially with n, e.g., O(n!), O(n^n), and O(2^n).
  - No known algorithms can ensure, in a time-efficient manner, globally optimal solution

- **Heuristic** algorithms are used to find near-optimal solutions
  - Be content with a “reasonably good” solution

Types of Algorithms

- There are many ways to categorize different types of algorithms
  - Polynomial vs. Exponential, in terms of computational effort
  - Optimal (or Exact) vs. Heuristic, in solution quality
  - Deterministic vs. Stochastic, in decision making
  - Constructive vs. Iterative, in structure

  …

Broader Classification of Algorithms

- Combinatorial algorithms
  - **Graph algorithms**
  - …
- Computational mathematics
  - **Optimization algorithms**
  - Numerical algorithms
  - …
- Computational science
  - Bioinformatics
  - Linguistics
  - Statistics
  - …
- Information theory & signal processing
- Many more

Topics touched in this course

[source: en.wikipedia.org/wiki/List_of_algorithms]

Graph Definition

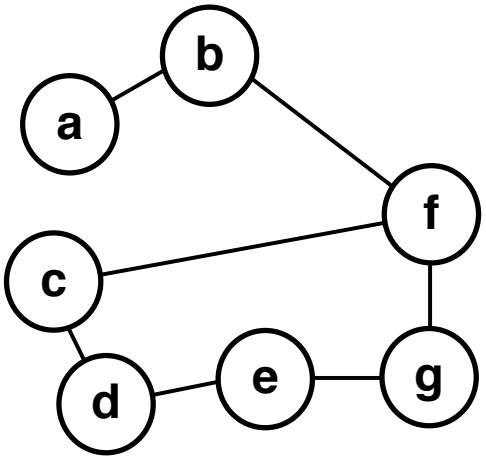
- Graph: a set of objects and their connections
  - Importance: any binary relation can be represented as a graph

- Formal definition:
  - $G = (V, E)$, $V = \{v_1, v_2, ..., v_n\}$, $E = \{e_1, e_2, ..., e_m\}$
    - V : set of **vertices** (nodes), E : set of **edges** (arcs)
  - **Undirected graph**: an edge {u, v} also implies {v, u}
  - **Directed graph**: each edge (u, v) has a direction

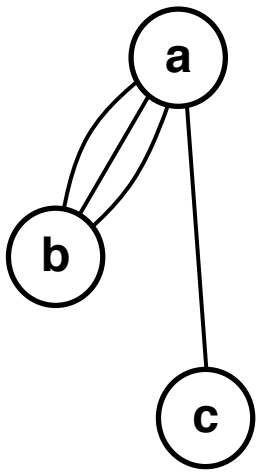
Simple Graph

- Loops, multi edges, and simple graphs
  - An edge of the form (v, v) is said to be a **self-loop**
  - A graph permitted to have multiple edges (or parallel edges) between two vertices is called a **multigraph**
  - A graph is said to be **simple** if it contains no self-loops or multiedges

Simple graph



Multigraph



Graph Connectivity

- Paths
  - A **path** is a sequence of edges connecting two vertices
  - A **simple path** never goes through any vertex more than once

- Connectivity
  - A graph is **connected** if there is there is a path between any two vertices
  - Any subgraph that is connected can be referred to as a **connected component**
  - A directed graph is **strongly connected** if there is always a directed path between vertices

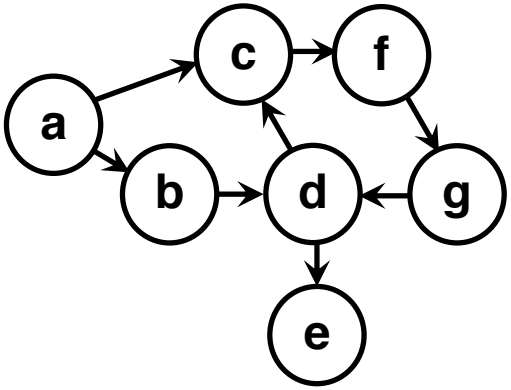
Trees and DAGs

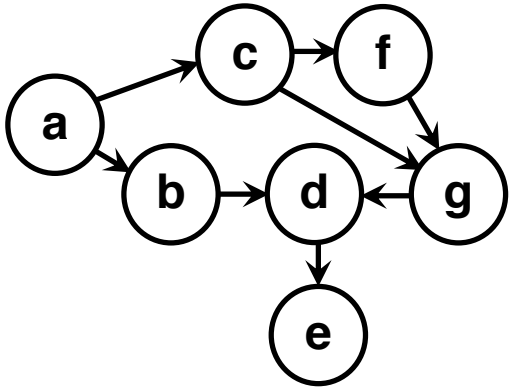
- A **cycle** is a path starting and ending at the same vertex. A cycle in which no vertex is repeated other than the starting vertex is said to be a **simple cycle**

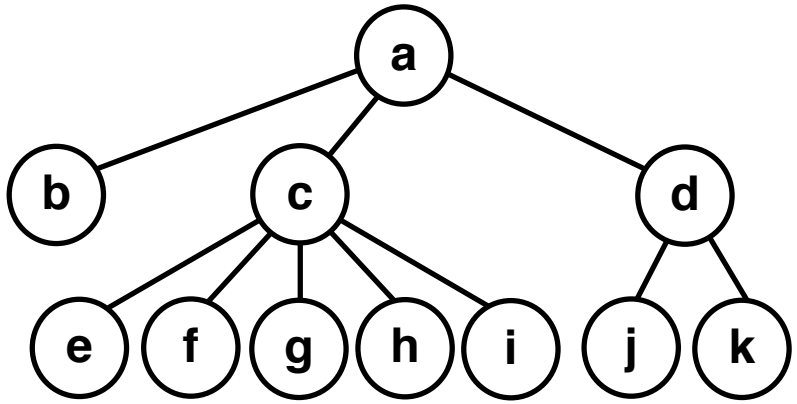
- An undirected graph with no cycles is a **tree** if it is connected, or a **forest** otherwise
  - A **directed tree** is a directed graph which would be a tree if the directions on the edges were ignored

- A directed graph with no directed cycles is said to be a **directed acyclic graph** (**DAG**)

Examples

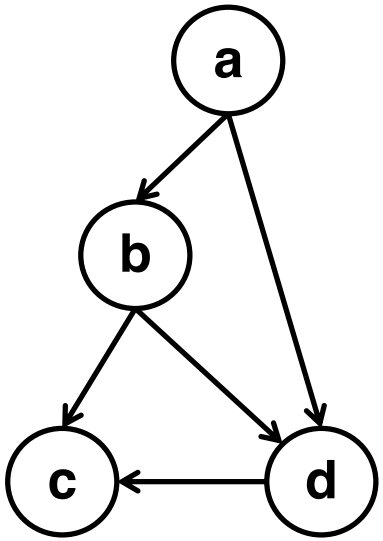


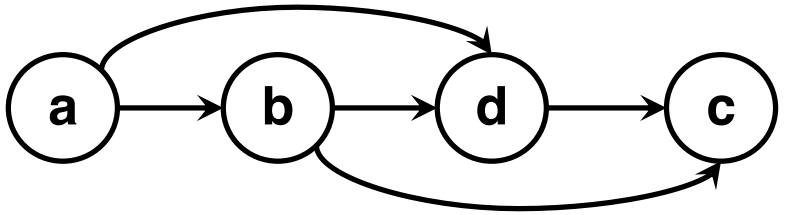




Topological Sort

- A **topological sort** (or order) of a directed graph is an ordering of nodes where all edges go from an earlier vertex (left) to a later vertex (right)
  - Feasible if and only if the subject graph is a DAG





Takeaway Points

- Exponential growth in silicon capacity calls for higher level of **design abstraction**
- End of Dennard scaling leads to increasing **specialization** to sustain improvement in hardware performance and energy efficiency
- EDA tools are fueled by highly sophisticated and yet scalable CAD **algorithms**

Before Next Lecture

- Actions
  - Check out the course website
  - Read through the course syllabus
  - **Verify your login on ecelinux**
    - ssh -X <netid>@ecelinux-01.ece.cornell.edu

HLS tutorial on Tuesday (10/19) at 4:40pm

Acknowledgements

- These slides contain/adapt materials developed by
  - Prof. Jason Cong (UCLA)
  - Prof. David Z. Pan (UT Austin)