



Fiscal Year 2020

Ver. 2020-12-02a

Course number: CSC.T433
School of Computing,
Graduate major in Computer Science

Advanced Computer Architecture

1. Design and Analysis of Computer Systems



www.arch.cs.titech.ac.jp/lecture/ACA/

Room No.W936

Mon 14:20-16:00, Thr 14:20-16:00

Kenji Kise, Department of Computer Science
kise_at_c.titech.ac.jp

Syllabus (1/3)



Course description and aims					
<p>This course aims to provide students with cutting-edge technologies and future trends of computer architecture with focusing on a microprocessor which plays an important role in the downsizing, personalization, and improvement of performance and power consumption of computer systems such as PCs, personal mobile devices, and embedded systems.</p> <p>In this course, first, along with important concepts of computer architecture, students will learn from instruction set architectures to mechanisms for extracting instruction level parallelism used in out-of-order superscalar processors. After that, students will learn mechanisms for exploiting thread level parallelism adopted in multi-processors and multi-core processors.</p>					
Student learning outcomes					
<p>By taking this course, students will learn:</p> <ul style="list-style-type: none">(1) Basic principles for building today's high-performance computer systems(2) Mechanisms for extracting instruction level parallelism used in high-performance microprocessors(3) Methods for exploiting thread level parallelism adopted in multi-processors and multi-core processors(4) New inter-relationship between software and hardware					
Keywords					
Computer Architecture, Processor, Embedded System, multi-processor, multi-core processor					
Competencies that will be developed					
<table><tr><td><input checked="" type="checkbox"/> Specialist skills</td><td>Intercultural skills</td><td>Communication skills</td><td>Critical thinking skills</td><td>Practical and/or problem-solving skills</td></tr></table>	<input checked="" type="checkbox"/> Specialist skills	Intercultural skills	Communication skills	Critical thinking skills	Practical and/or problem-solving skills
<input checked="" type="checkbox"/> Specialist skills	Intercultural skills	Communication skills	Critical thinking skills	Practical and/or problem-solving skills	
Class flow					
<p>Before coming to class, students should read the course schedule and check what topics will be covered. Required learning should be completed outside of the classroom for preparation and review purposes.</p>					

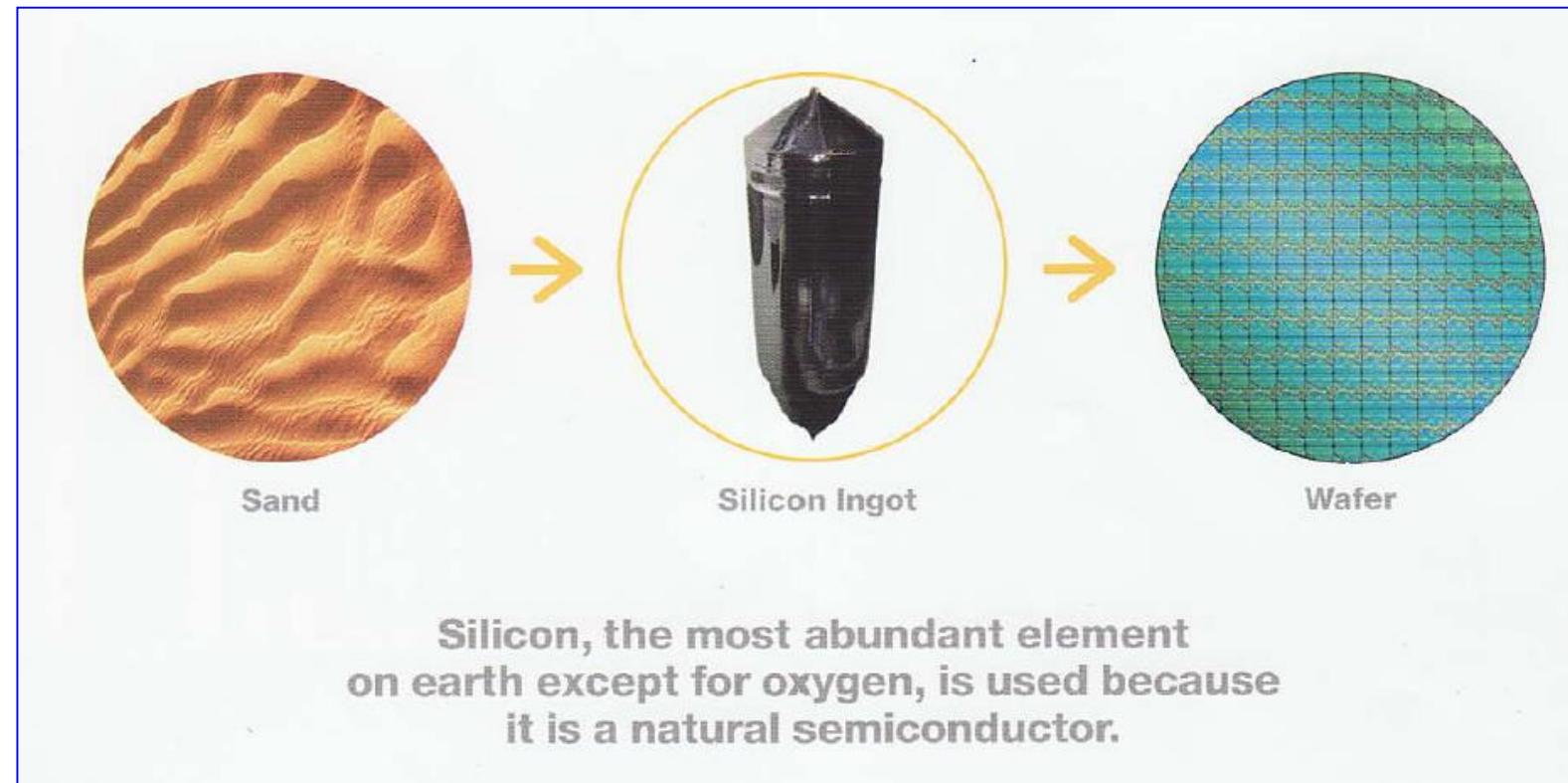
<http://www.ocw.titech.ac.jp/index.php?module=General&action=T0300&GakubuCD=4&GakkaCD=342323&KeiCD=23&course=23&KougiCD=202004864&Nendo=2020&lang=EN&vid=03>



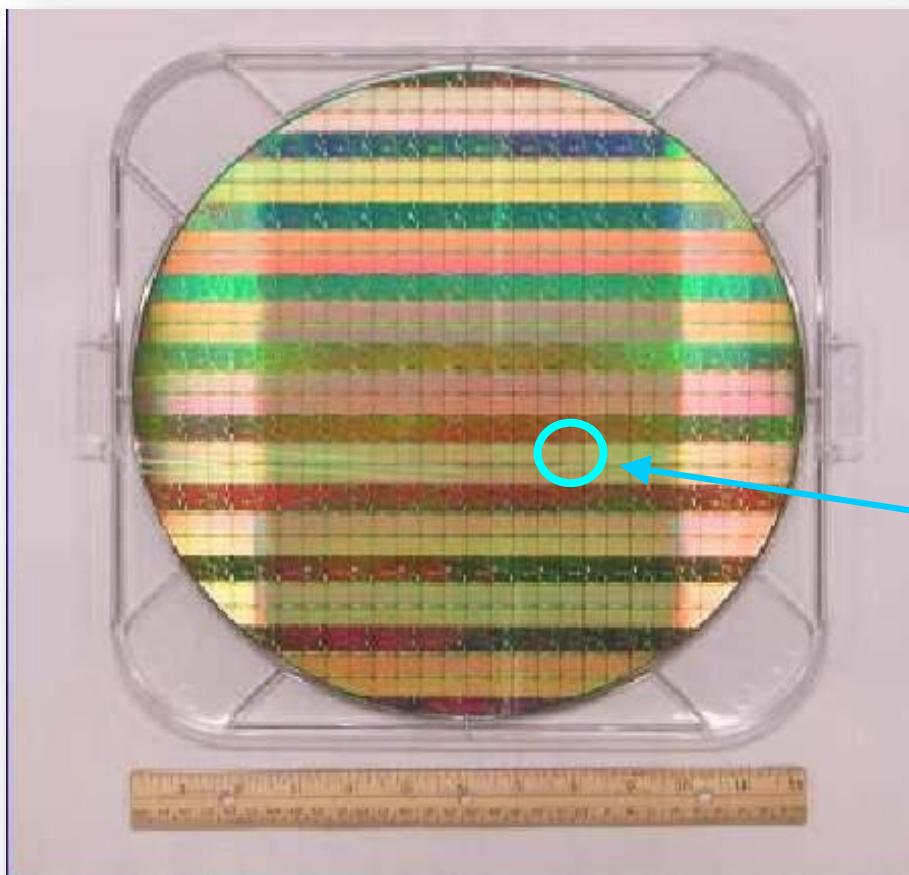
Syllabus (2/3)

Textbook(s)
John L. Hennessy, David A. Patterson. Computer Architecture A Quantitative Approach, Fifth Edition. Morgan Kaufmann Publishers Inc., 2012
Reference books, course materials, etc.
William James Dally, Brian Patrick Towles. Principles and Practices of Interconnection Networks. Morgan Kaufman Publishers Inc., 2004.
Assessment criteria and methods
Students will be assessed on their understanding of instruction level parallelism, multi-processor, and thread level parallelism. Students' course scores are based on the mid-term report and assignments (40%), and the final report (60%).
Related courses
CSC.T363 : Computer Architecture CSC.T341 : Computer Logic Design
Prerequisites (i.e., required knowledge, skills, courses, etc.)
No prerequisites are necessary, but enrollment in the related courses is desirable.
Contact information (e-mail and phone) Notice : Please replace from "[at]" to "@"(half-width character).
Kise Kenji: kise[at]c.titech.ac.jp
Office hours
Contact by e-mail in advance to schedule an appointment.

Processor fabrication: **ingot** and **wafer**



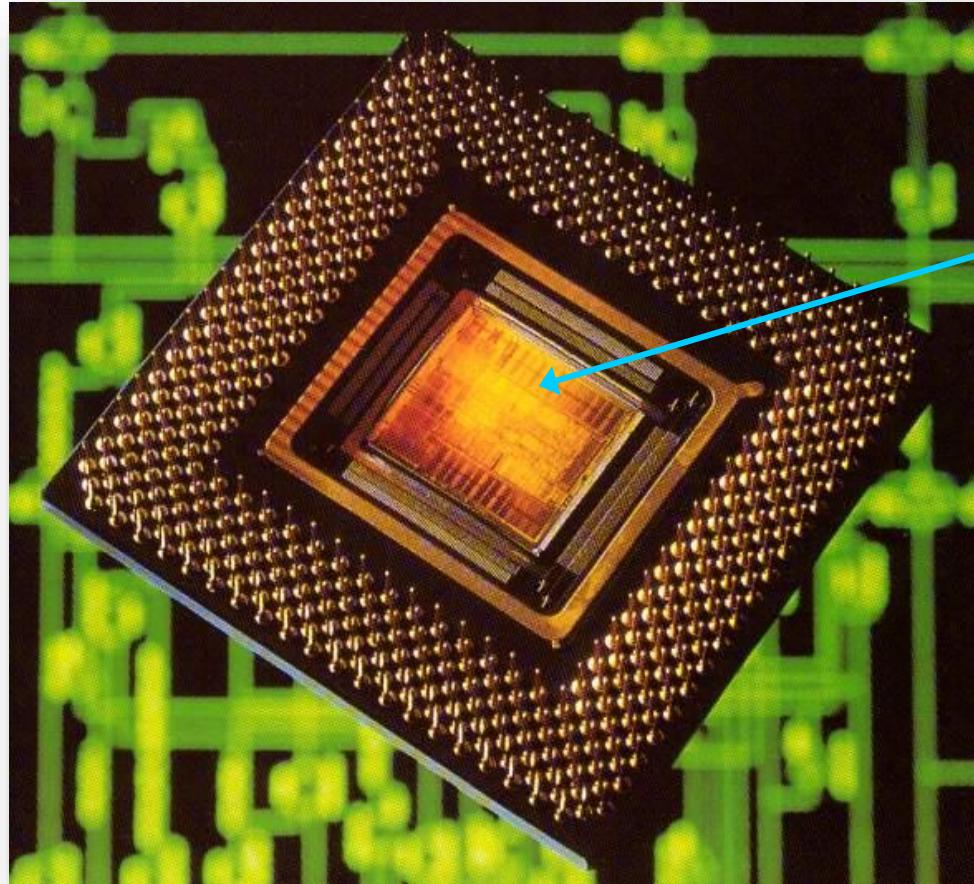
Processor fabrication: wafer and die



Die

Intel, Industry-Leading Transistor Performance Demonstrated on Intel's 90-nanometer Logic Process

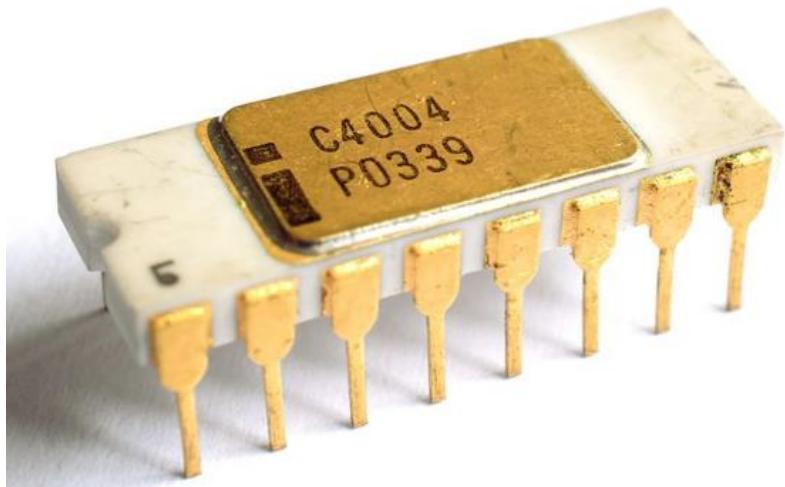
Processor fabrication: die and packaging



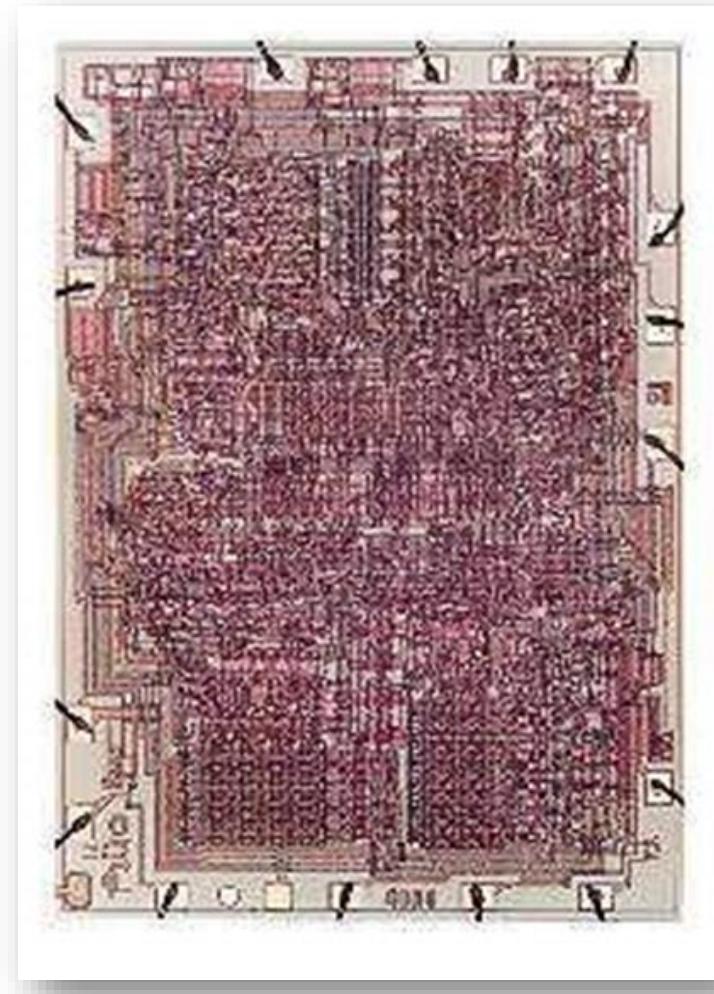
Die



The birth of microprocessors



Name	Year	# of transistors
Intel 4004	1971	2,250



Moore's Law

- Moore's law is the observation that the number of transistors in a dense integrated circuit doubles about every two years. The observation is named after Gordon Moore, the co-founder of Fairchild Semiconductor and Intel, whose 1965 paper described a doubling every year in the number of components per integrated circuit, and projected this rate of growth would continue for at least another decade. In 1975, looking forward to the next decade, he revised the forecast to doubling every two years. The period is often quoted as 18 months because of a prediction by Intel executive David House (being a combination of the effect of more transistors and the transistors being faster).



Moore's Law

VISUALIZING PROGRESS

If transistors were people

If the transistors in a microprocessor were represented by people, the following timeline gives an idea of the pace of Moore's Law.



2,300
Average music hall capacity



134,000
Large stadium capacity



32 Million
Population of Tokyo



1.3 Billion
Population of China

1971
Intel 4004

1981
Intel 286

1990

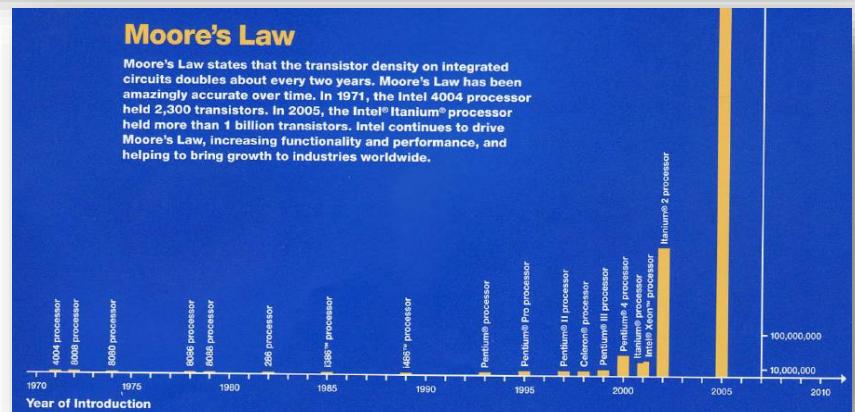
2000

Pentium III

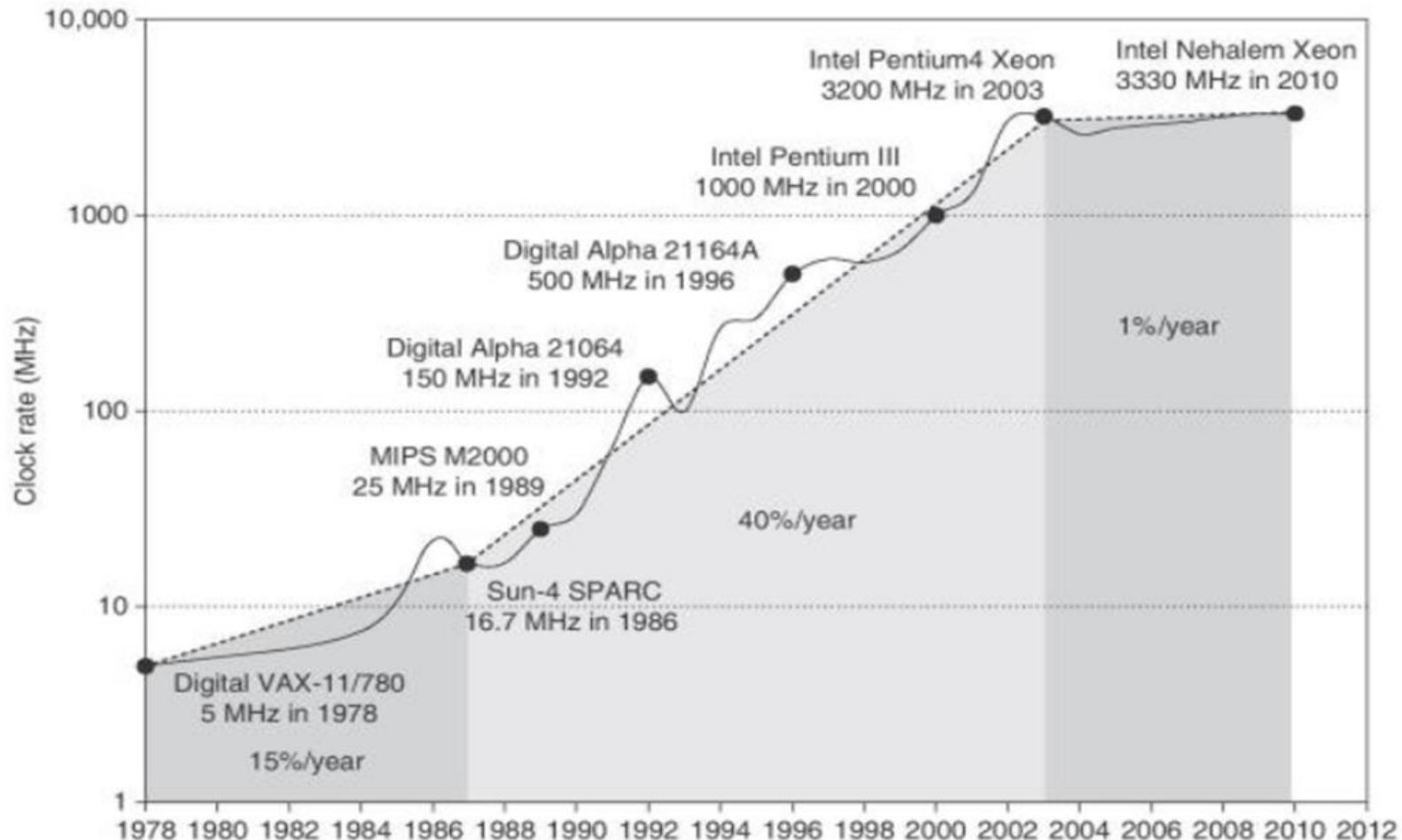
2011

Core i7 Extreme Edition

Now imagine that those 1.3 billion people could fit onstage in the original music hall. That's the scale of Moore's Law.



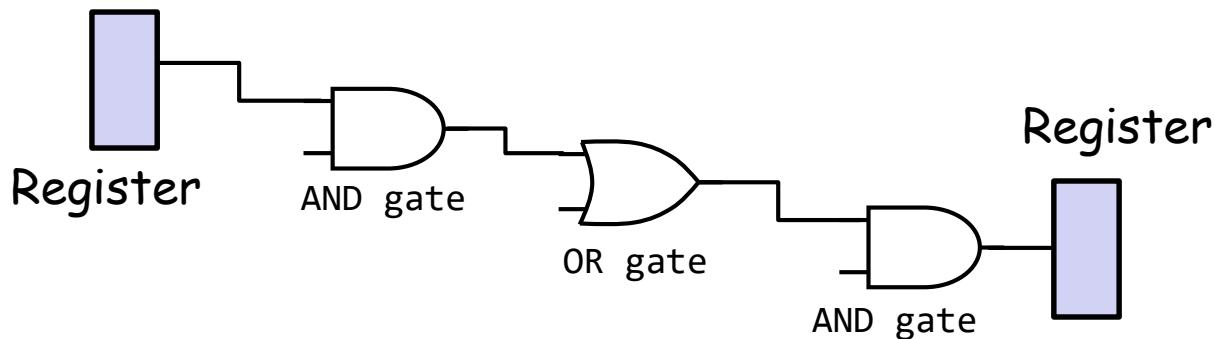
Growth in clock rate of microprocessors

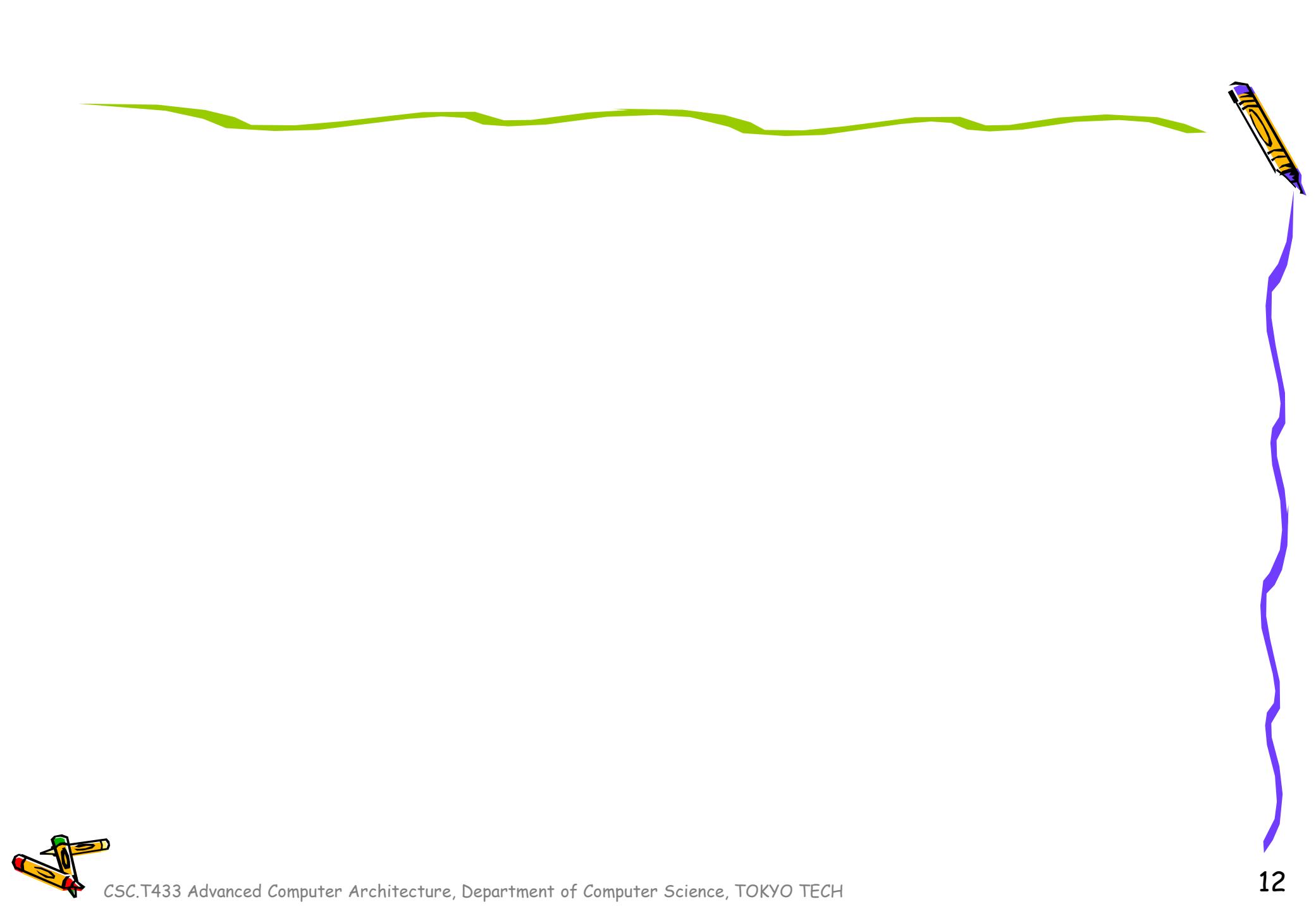


From CAQA 5th edition

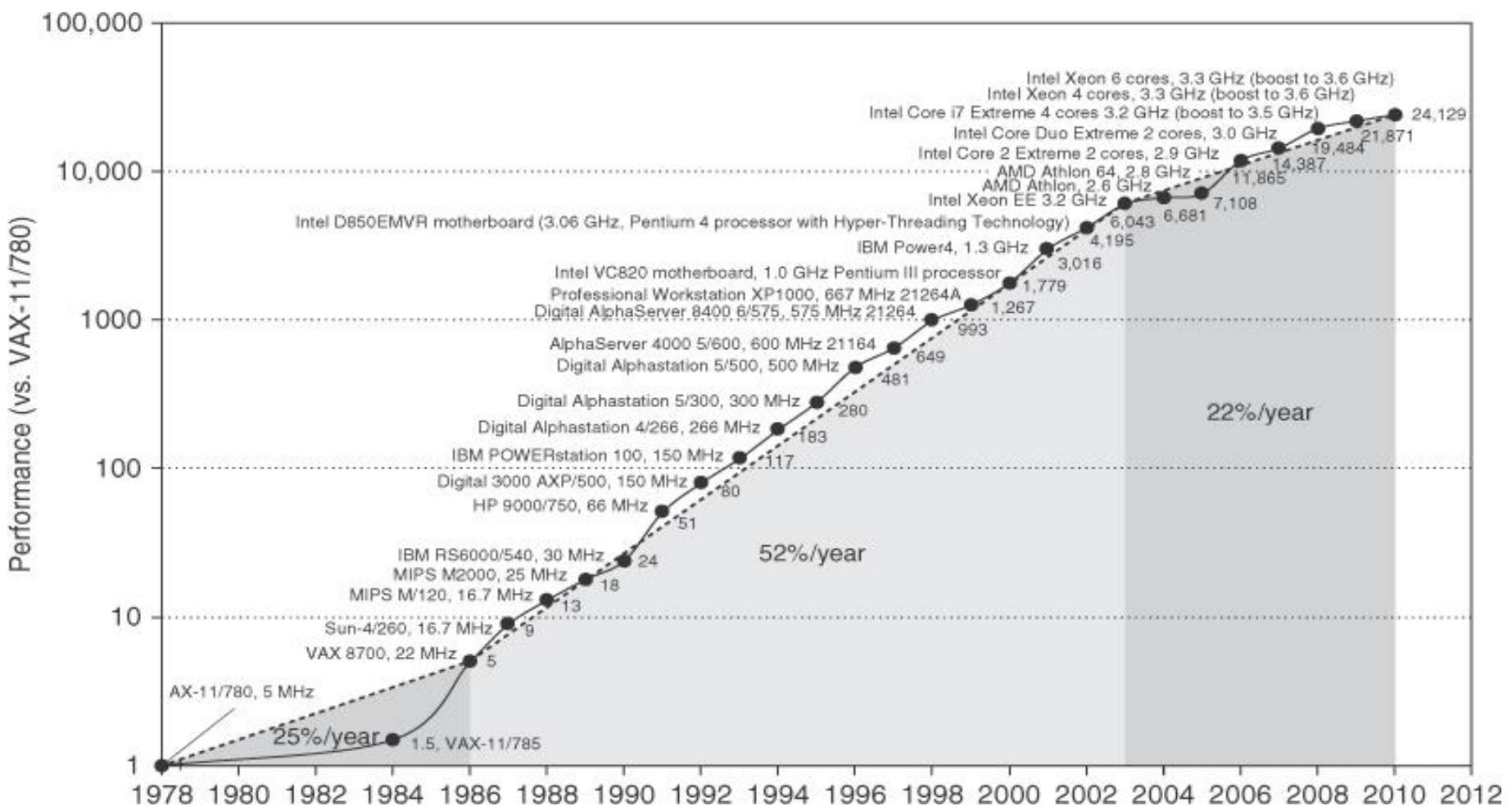
Clock rate is mainly determined by

- Switching speed of gates (transistors)
- The number of levels of gates
 - The maximum number of gates cascaded in series in any combinational logics.
 - In this example, the number of levels of gates is 3.
- Wiring delay and fanout





Growth in processor performance



From CAQA 5th edition

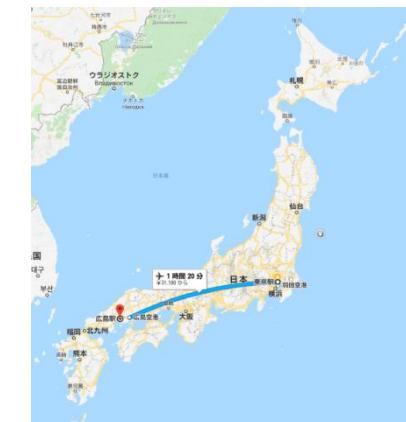
Which is faster?

From Tokyo to Hiroshima



	Time & Cost	Max Speed	Passengers	Throughput (Speed x P)
Boeing 737	1:20 32,000yen	800km/h (503km/h)	170	85,510 (503 x 170)
Nozomi	4:00 18,000yen	270km/h (205km/h)	1,300	266,500 (205 x 1,300)

- Time to run the task (ExTime)
 - Execution time, response time, latency
- Tasks per day, hour, week, sec, ns ... (Performance)
 - Throughput, bandwidth



Based on the lecture slide of David E Culler

CSC.T433 Advanced Computer Architecture, Department of Computer Science, TOKYO TECH

Defining (Speed) Performance

- Normally interested in reducing
 - **Response time** (execution time) – the time between the start and the completion of a **task** or a **program**
 - Important to individual users
 - Thus, **to maximize performance, need to minimize execution time**

$$\text{performance}_x = 1 / \text{execution_time}_x$$

If X is n times faster than Y, then

$$\frac{\text{performance}_x}{\text{performance}_y} = \frac{\text{execution_time}_y}{\text{execution_time}_x} = n$$

- Throughput – the total amount of work done in a given time
 - Important to data center managers
- Decreasing response time almost always improves throughput



Performance Factors

- Want to distinguish elapsed time and the time spent on our task
- CPU execution time (CPU time) : time the CPU spends working on a task
 - Does not include time waiting for I/O or running other programs

$$\text{CPU execution time for a program} = \frac{\text{\# CPU clock cycles for a program}}{\text{clock cycle time}}$$

or

$$\text{CPU execution time for a program} = \frac{\text{\# CPU clock cycles for a program}}{\text{clock rate}}$$

- Can improve performance by reducing either the **length of the clock cycle** or the **number of clock cycles required for a program**



Performance Factors

$$\text{CPU execution time for a program} = \frac{\text{\# CPU clock cycles for a program}}{\text{clock rate}}$$

$$\text{Performance for a program} = \text{clock rate} \times 1 / \text{\# CPU clock cycles for a program}$$

- $\text{Performance} = f \times \text{IPC}$
 - f : frequency (clock rate)
 - IPC: retired instructions per cycle

```
int flag = 1;  
  
int foo(){  
    while(flag);  
}
```

Syllabus (3/3)

Course schedule/Required learning		
	Course schedule	Required learning
Class 1	Design and Analysis of Computer Systems	Understand the basic of design and analysis of computer systems.
Class 2	Instruction Set Architecture	Understand the examples of instruction set architectures
Class 3	Memory Hierarchy Design	Understand the organization of memory hierarchy designs
Class 4	Pipelining	Understand the idea and organization of pipelining
Class 5	Instruction Level Parallelism: Concepts and Challenges	Understand the idea and requirements for exploiting instruction level parallelism
Class 6	Instruction Level Parallelism: Instruction Fetch and Branch Prediction	Understand the organization of instruction fetch and branch predictions to exploit instruction level parallelism
Class 7	Instruction Level Parallelism: Advanced Techniques for Branch Prediction	Understand the advanced techniques for branch prediction to exploit instruction level parallelism
Class 8	Instruction Level Parallelism: Dynamic Scheduling	Understand the dynamic scheduling to exploit instruction level parallelism
Class 9	Instruction Level Parallelism: Exploiting ILP Using Multiple Issue and Speculation	Understand the multiple issue mechanism and speculation to exploit instruction level parallelism
Class 10	Instruction Level Parallelism: Out-of-order Execution and Multithreading	Understand the out-of-order execution and multithreading to exploit instruction level parallelism



From multi-core era to many-core era

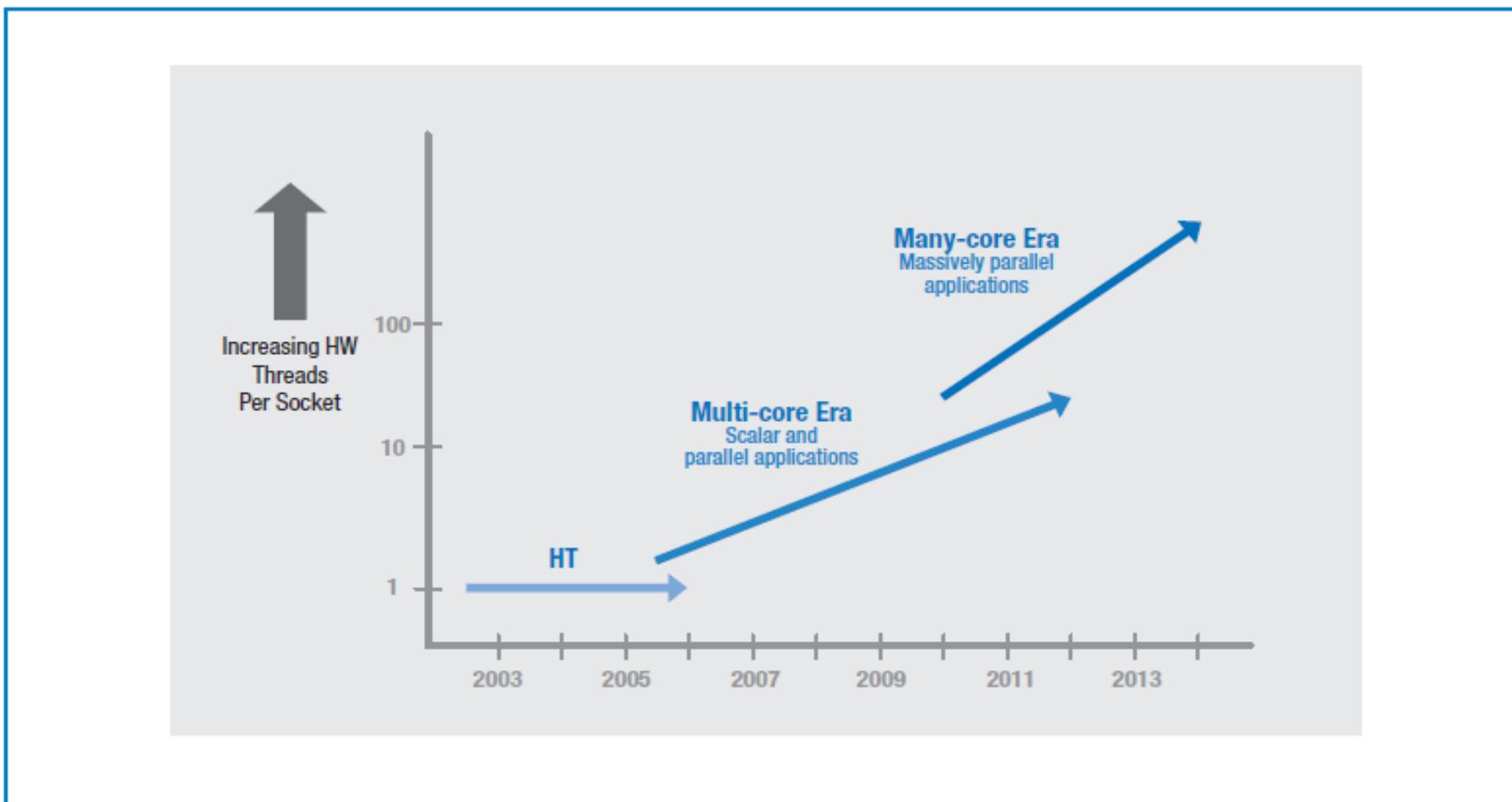


Figure 1: Current and expected eras of Intel® processor architectures

Platform 2015: Intel® Processor and Platform Evolution for the Next Decade, 2005

Pollack's Rule

- Pollack's Rule states that microprocessor "performance increase due to microarchitecture advances is roughly proportional to the square root of the increase in complexity". Complexity in this context means processor logic, i.e. its area.

WIKIPEDIA

From multi-core era to many-core era

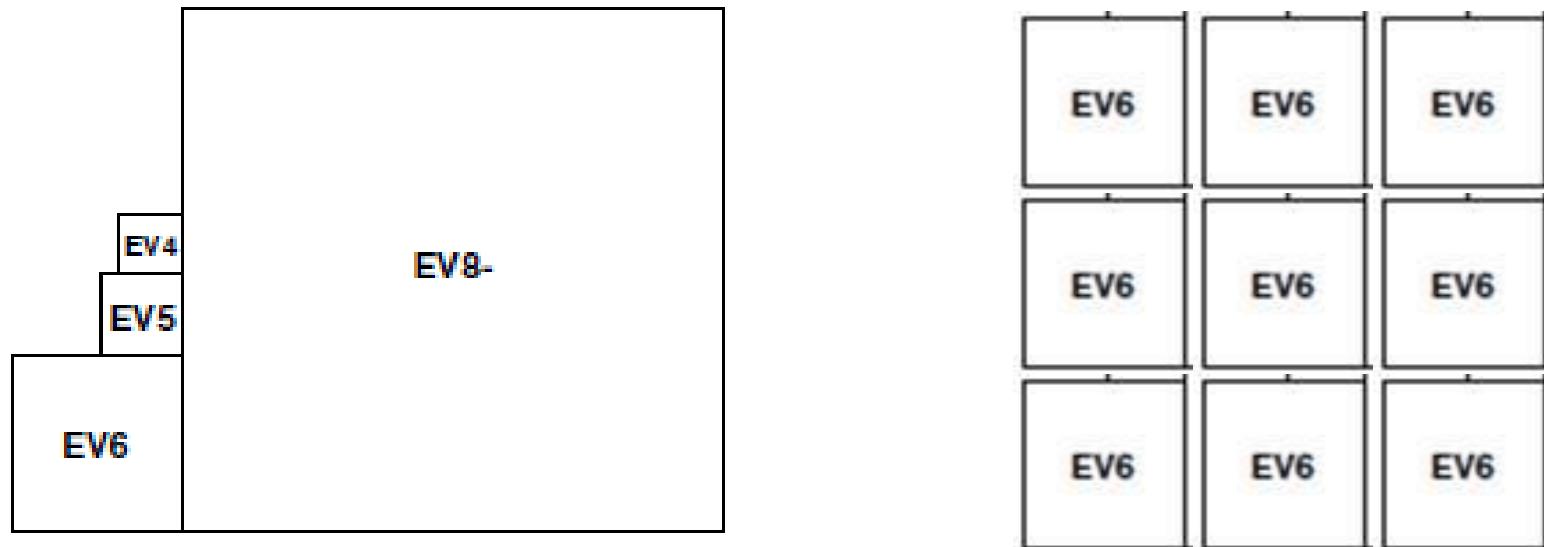
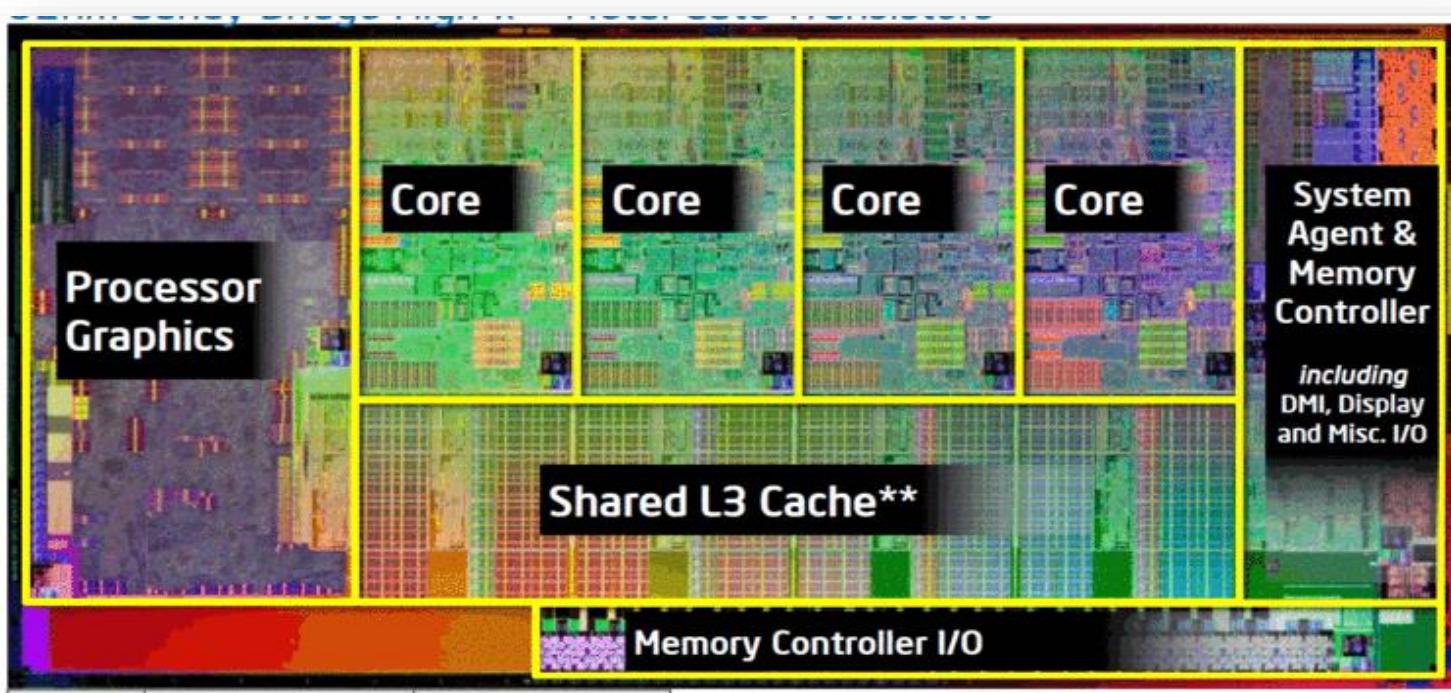


Figure 1. Relative sizes of the cores used in the study

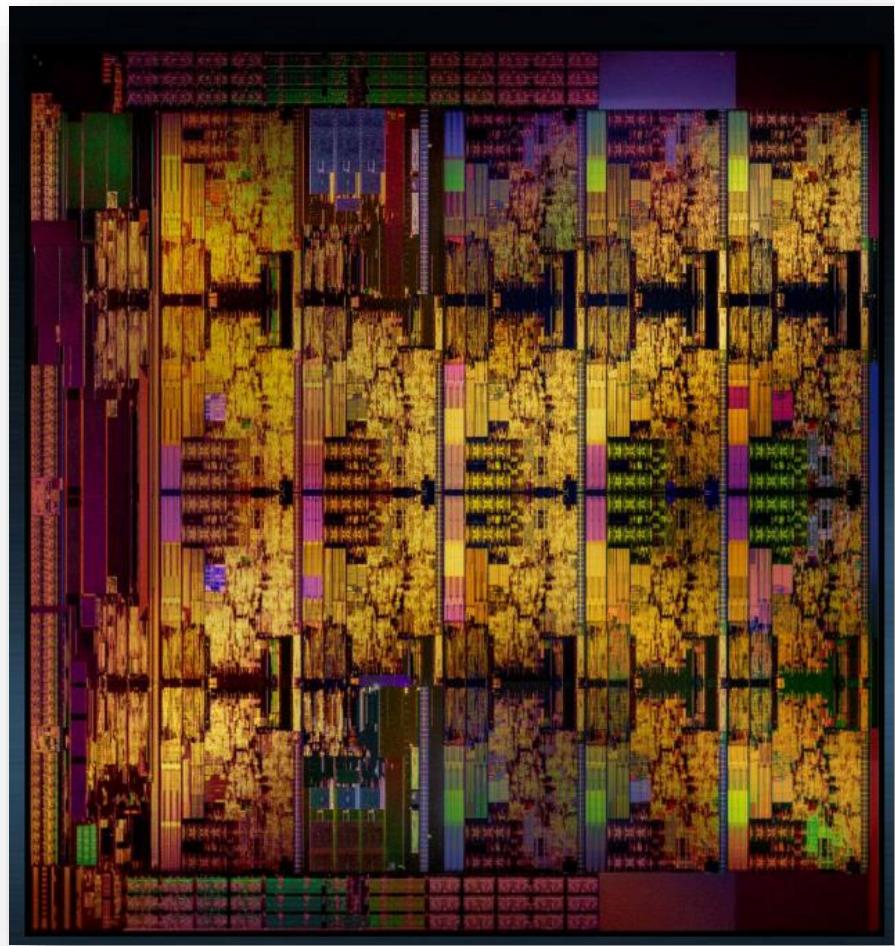
Intel Sandy Bridge, January 2011

- 4 core

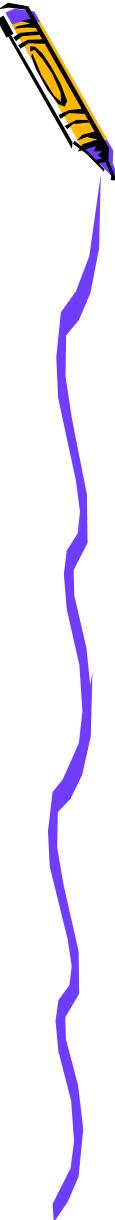


Intel Skylake-X, Core i9-7980XE, 2017

- 18 core



Syllabus (3/3)



Course schedule/Required learning		
	Course schedule	Required learning
Class 1	Design and Analysis of Computer Systems	Understand the basic of design and analysis of computer systems.
Class 2	Instruction Set Architecture	Understand the examples of instruction set architectures
Class 3	Memory Hierarchy Design	Understand the organization of memory hierarchy designs
Class 4	Pipelining	Understand the idea and organization of pipelining
Class 5	Instruction Level Parallelism: Concepts and Challenges	Understand the idea and requirements for exploiting instruction level parallelism
Class 6	Instruction Level Parallelism: Instruction Fetch and Branch Prediction	Understand the organization of instruction fetch and branch predictions to exploit instruction level parallelism
Class 7	Instruction Level Parallelism: Advanced Techniques for Branch Prediction	Understand the advanced techniques for branch prediction to exploit instruction level parallelism
Class 8	Instruction Level Parallelism: Dynamic Scheduling	Understand the dynamic scheduling to exploit instruction level parallelism
Class 9	Instruction Level Parallelism: Exploiting ILP Using Multiple Issue and Speculation	Understand the multiple issue mechanism and speculation to exploit instruction level parallelism
Class 10	Instruction Level Parallelism: Out-of-order Execution and Multithreading	Understand the out-of-order execution and multithreading to exploit instruction level parallelism
Class 11	Multi-Processor: Distributed Memory and Shared Memory Architecture	Understand the distributed memory and shared memory architecture for multi-processors
Class 12	Thread Level Parallelism: Coherence and Synchronization	Understand the coherence and synchronization for thread level parallelism
Class 13	Thread Level Parallelism: Memory Consistency Model	Understand the memory consistency model for thread level parallelism
Class 14	Thread Level Parallelism: Interconnection Network and Many-core Processors	Understand the interconnection network and many-core processors for thread level parallelism



Adaptive Computing Research Initiative (ACRI)

- The aim
 - Aiming to develop the high-performance Adaptive Computing Systems that utilize FPGAs
 - Working out to distribute the FPGA-related technologies, including our developed systems, as an outreach activity for research results
- Main research theme
 1. Development for FPGA accelerator to speed up processing of AI etc.
 2. Development for FPGA accelerators and FPGA systems for IoT.
- Activity
 - Establishment Date: April 1st 2020
 - Activity period : First period 3 years



The Adaptive Computing Research Initiative is an organization to seek out and research ways to utilize FPGAs.



Please apply for your user account on this site **today**

- <https://gw.acri.c.titech.ac.jp/wp/manual/apply-for-account>

ACRI ルームのアカウント申請方法

2020.08.11 2020.06.30

□ 目次 [閉じる]

1. アカウントの申請
 - 登録フォームへの入力
 - アカウント申請の受理
2. ログインおよびパスフレーズの変更
 - 予約システム
 - ACRI のサーバ

アカウントの申請

登録フォームへの入力

申請するには、「アカウントの申請」のリンクをクリックするか、ログインが必要なページ(例えば各サーバの予約状況のページ)にアクセスしてください。後者の場合には、ログインフォームの右下にある「新規ユーザー登録」をクリックしてください。

ACRI ルームの情報

ようこそ
予約ページトップ
ニュースとメンテナンス情報
フォーラム
ギャラリーと技術情報

ログイン/ログアウト

ログイン

ACRI ルームの利用説明

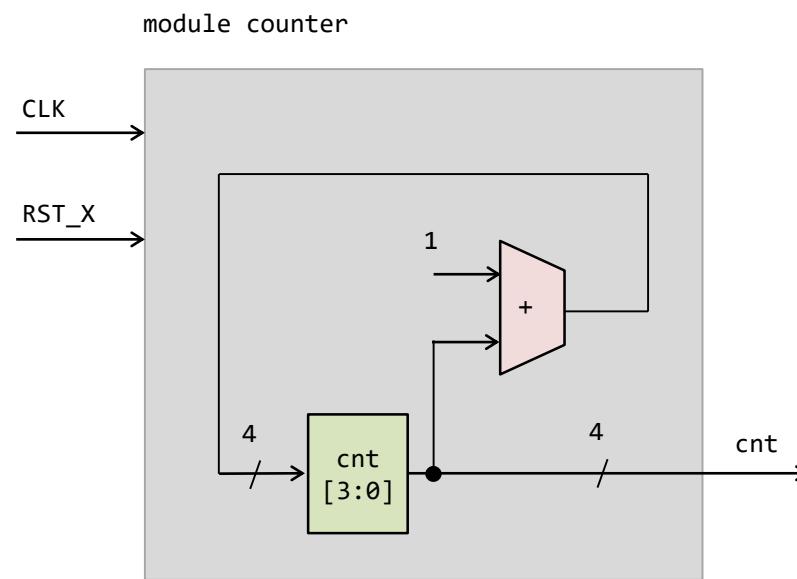
利用規約



Sample circuit 1

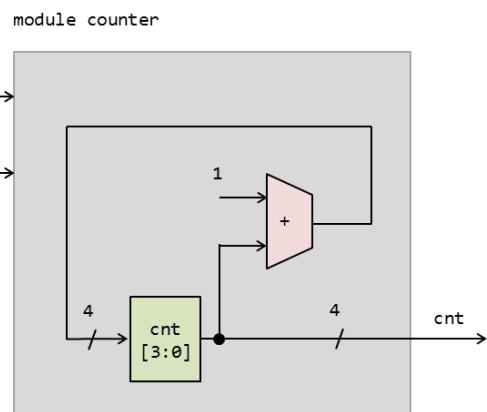
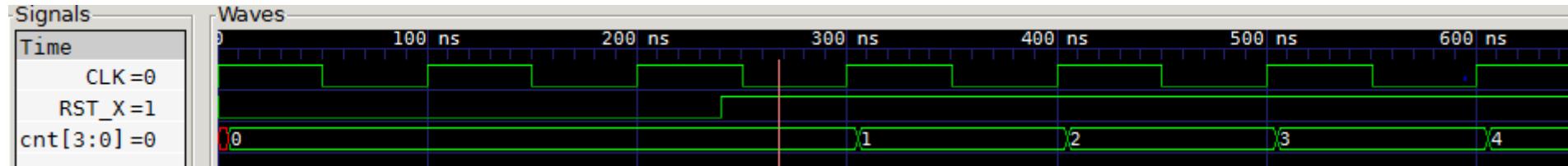


- 4-bit counter
 - synchronous reset
 - negative-logic reset, initialize or reset the value of register cnt to zero if RST_X is low





Sample Verilog HDL Code



counter.v

```
8 module top();
9   reg CLK, RST_X;
10  wire [3:0] w_cnt;
11
12 initial begin CLK = 1; forever #50 CLK = ~CLK; end
13 initial begin RST_X = 0; #240 RST_X = 1; end
14 initial #800 $finish();
15 initial begin
16   $dumpfile("wave.vcd");
17   $dumpvars(0, cnt1);
18 end
19 always @(posedge CLK) $write("cnt1: %d %x\n", RST_X, w_cnt);
20
21 counter cnt1(CLK, RST_X, w_cnt);
22 endmodule
23
24 ****
25 module counter(CLK, RST_X, cnt);
26   input wire CLK, RST_X;
27   output reg [3:0] cnt;
28
29   always @(posedge CLK) begin
30     if(!RST_X) cnt <= #5 0;
31     else      cnt <= #5 cnt + 1;
32   end
33 endmodule
```

