Embedded Intelligent System and Novel Computer Architecture

Lecture 02 – Key themes of Parallel computing

Pengju Ren
Institute of Artificial Intelligence and Robotics
Xi'an Jiaotong University

http://gr.xjtu.edu.cn/web/pengjuren

Programmer's Perspective on Performance

Question: How do you make your program run faster?

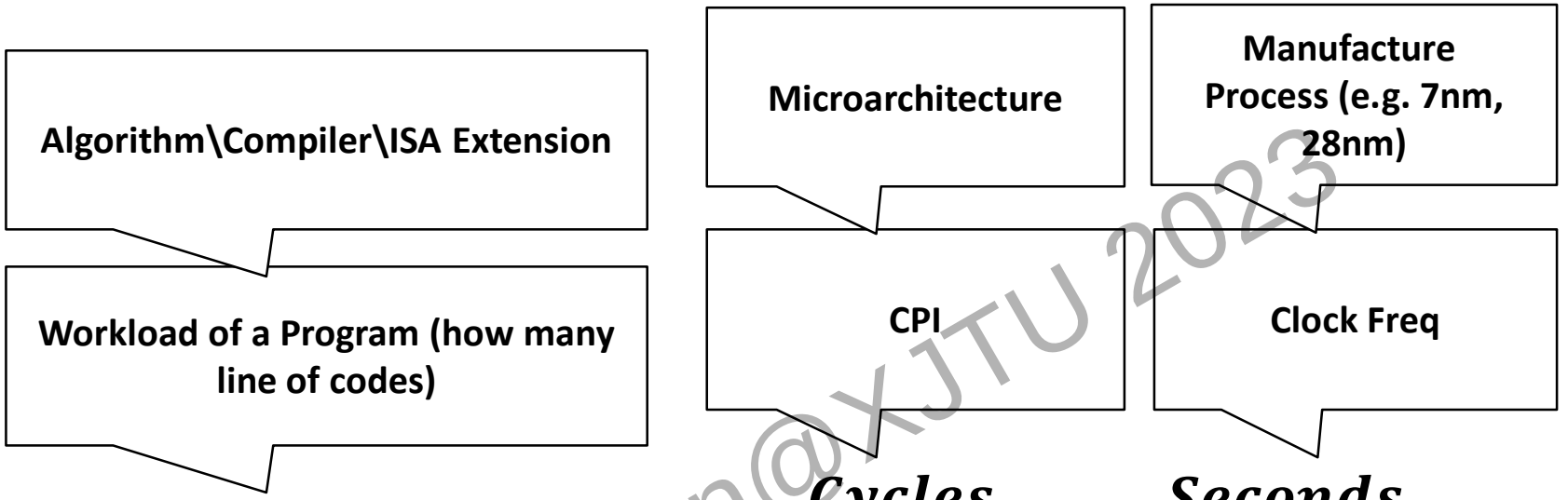
Answer before 2004:

- **Just wait 6 months, and buy a new machine!**
- **(Or if you're really obsessed, you can learn about parallelism.)**

Answer after 2004:

- **You need to learn the underlying parallel hardware architecture and write better parallel software.**

“Iron Law” of uniprocessor performance



$$Total\ Instructions \times \frac{Cycles}{Instruction} \times \frac{Seconds}{Cycle}$$

- Instructions per program depends on Task, Algorithm, source code, compiler technology, and ISA
- Cycles per instructions (CPI) depends on ISA and microarchitecture (is a workload on average)
- Time per cycle depends upon the microarchitecture and base technology (28nm, 7nm, 3nm)

A Brief History of Processor Performance

- Wider data paths
  - 4 bit → 8 bit → 16 bit → 32 bit → 64 bit
- More efficient pipelining
  - e.g., 3.5 Cycles Per Instruction (CPI) → 1.1 CPI
- Exploiting instruction-level parallelism (ILP) (Next lecture)
  - "Superscalar" processing: e.g., issue up to 4 instructions/cycle
  - "Out-of-order" processing: extract parallelism from instruction stream
- Faster clock rates
  - e.g., 10 MHz → 200 MHz → 3 GHz
- During the 80s and 90s: large exponential performance gains
  - and then…

ILP tapped out + end of frequency scaling

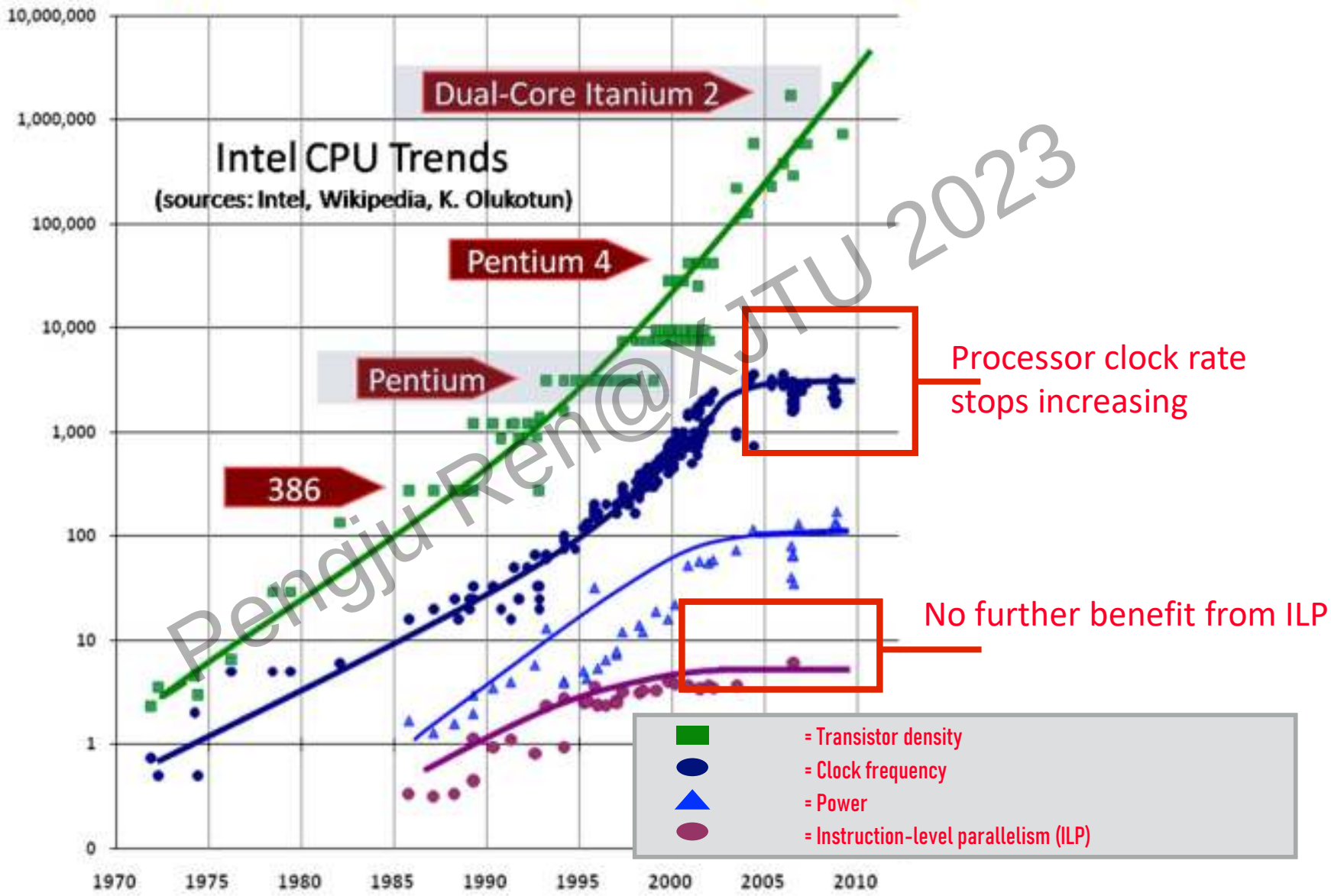


Image credit: “The free Lunch is Over” by Herb Sutter, Dr. Dobbs 2005

The “power wall”

Power consumed by a transistor:

Dynamic power ∝ capacitive load × voltage^2 × frequency

Static power: transistors burn power even when inactive

High power = high heat

Power is a critical design constraint in modern processors

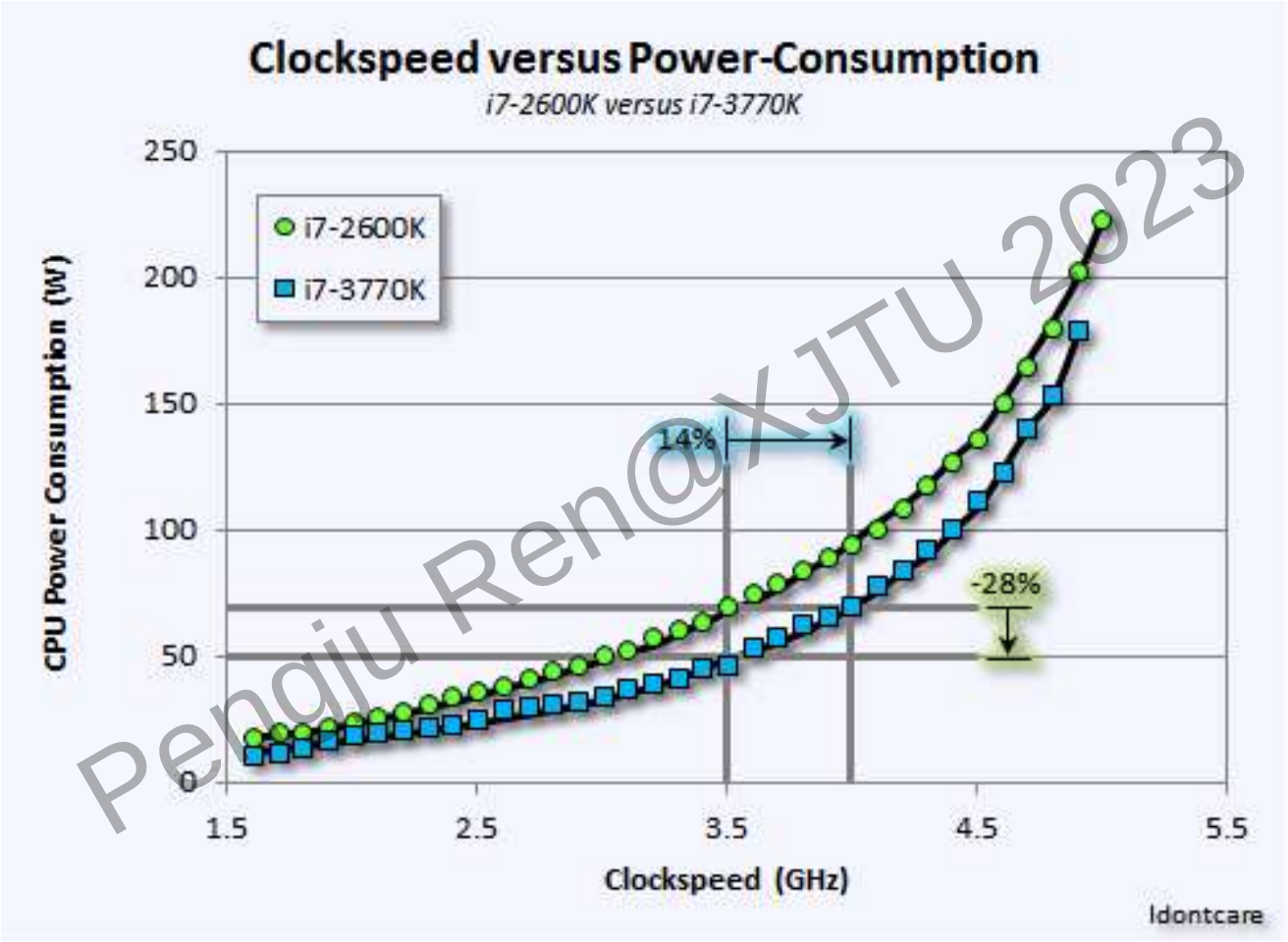
Air Cooling

Liquid Cooling

TPU-V4

NV4090

Power draw as a function of frequency

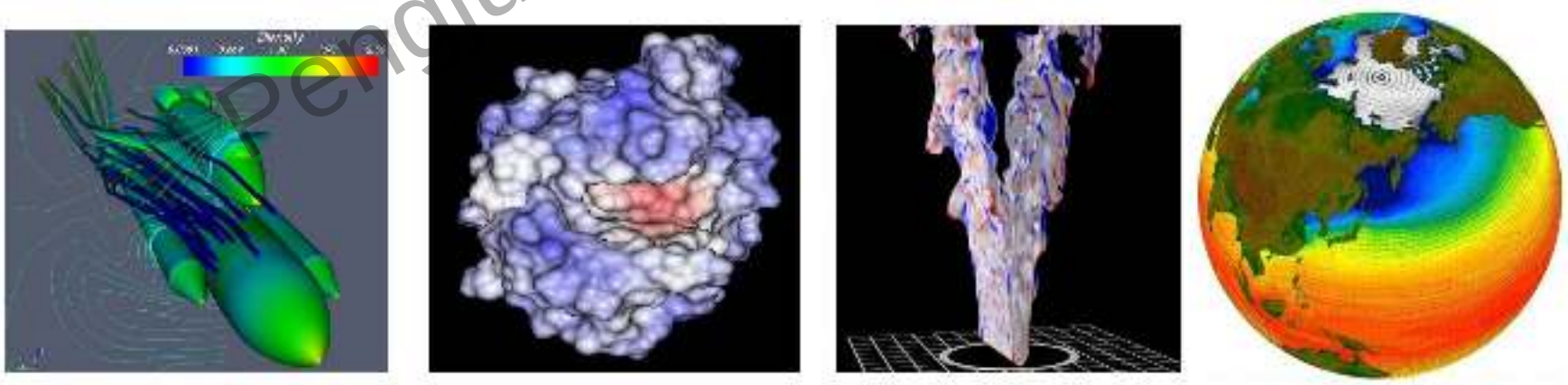


Maximum allowed frequency determined by processor's core voltage

Review: Why we need parallel computing?

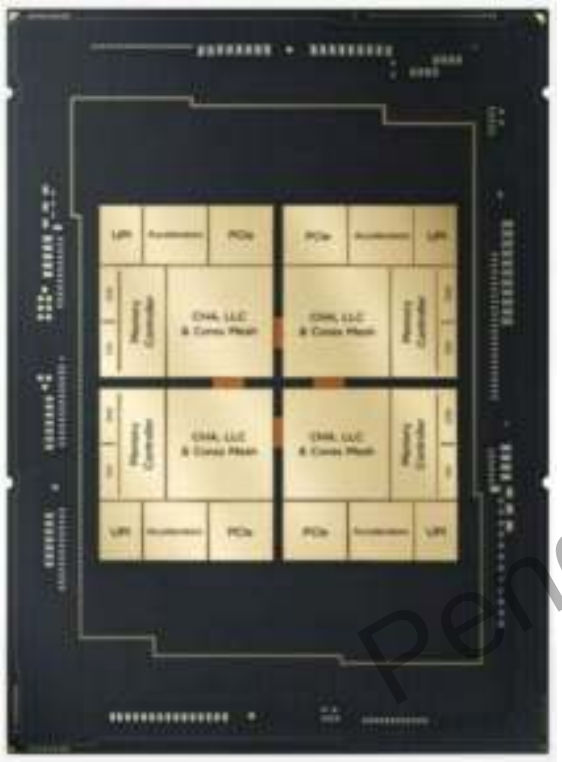
Recap: why parallelism ? (App driven)

- **Save time & money**
  - Use more resources to shorten execution with potential cost saving
- **Solve larger problem**
  - Impossible or impractical to solve on a single computer
  - Scientific computing: simulation for weapon, medicine, climate …
- **Processing at the edge**
  - Autonomous agents, e.g Drone, AMR, Robotics



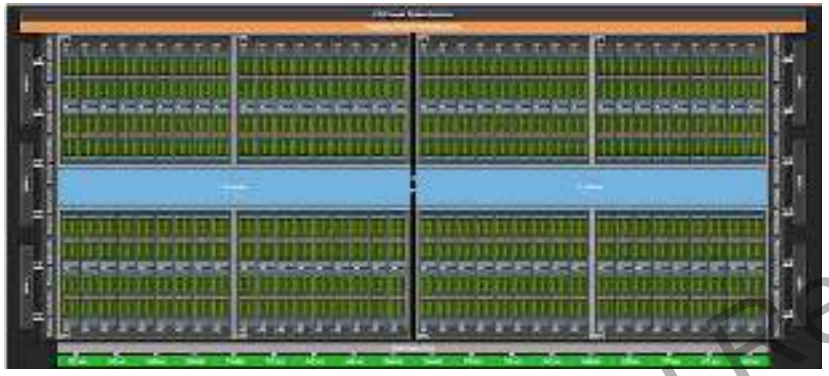
Peta-Byte(10^15) FLOPS, TB data

Server and Cloud: Intel Xeon Sapphire Rapids (2022)



8/16~72/144 (#cores/#threads)

Server and Cloud:NVIDIA H100 GPU@4nm (2022)



NVIDIA Accelerator Specification Comparison				
	H100 SXM	H100 PCIe	A100 SXM	A100 PCIe
FP32 CUDA Cores	16896	14592	6912	6912
Tensor Cores	528	456	432	432
Boost Clock	~1.78GHz (Not Finalized)	~1.65GHz (Not Finalized)	1.41GHz	1.41GHz
Memory Clock	4.8Gbps HBM3	3.2Gbps HBM2e	3.2Gbps HBM2e	3.0Gbps HBM2e
Memory Bus Width	5120-bit	5120-bit	5120-bit	5120-bit
Memory Bandwidth	3TB/sec	2TB/sec	2TB/sec	2TB/sec
VRAM	80GB	80GB	80GB	80GB
FP32 Vector	60 TFLOPS	48 TFLOPS	19.5 TFLOPS	19.5 TFLOPS
FP64 Vector	30 TFLOPS	24 TFLOPS	9.7 TFLOPS (1/2 FP32 rate)	9.7 TFLOPS (1/2 FP32 rate)
INT8 Tensor	2000 TOPS	1600 TOPS	624 TOPS	624 TOPS
FP16 Tensor	1000 TFLOPS	800 TFLOPS	312 TFLOPS	312 TFLOPS
TF32 Tensor	500 TFLOPS	400 TFLOPS	156 TFLOPS	156 TFLOPS
FP64 Tensor	60 TFLOPS	48 TFLOPS	19.5 TFLOPS	19.5 TFLOPS
Interconnect	NVLink 4 18 Links (900GB/sec)	NVLink 4 (600GB/sec)	NVLink 3 12 Links (600GB/sec)	NVLink 3 12 Links (600GB/sec)
GPU	GH100 (814mm2)	GH100 (814mm2)	GA100 (826mm2)	GA100 (826mm2)
Transistor Count	80B	80B	54.2B	54.2B
TDP	700W	350W	400W	300W
Manufacturing Process	TSMC 4N	TSMC 4N	TSMC 7N	TSMC 7N

Embedded Systems@Edge

Apple A15@5nm: (in iPhone 13pro)

6 CPU cores@3.23GHz

5 GPU cores

Bionic core (16 core NPU)

+much more

NVIDIA Jetson Nano

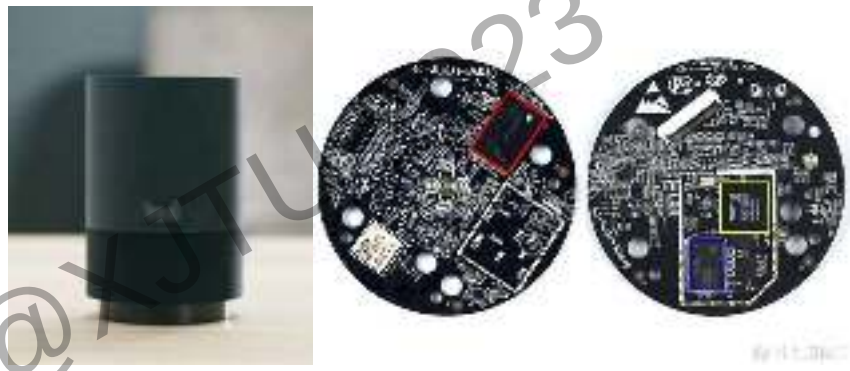
4 core ARM Cortex-A57

128 NV CUDA cores

Embedded Systems@Edge

Respberry Pi 3

Quad-core ARM A53 CPU



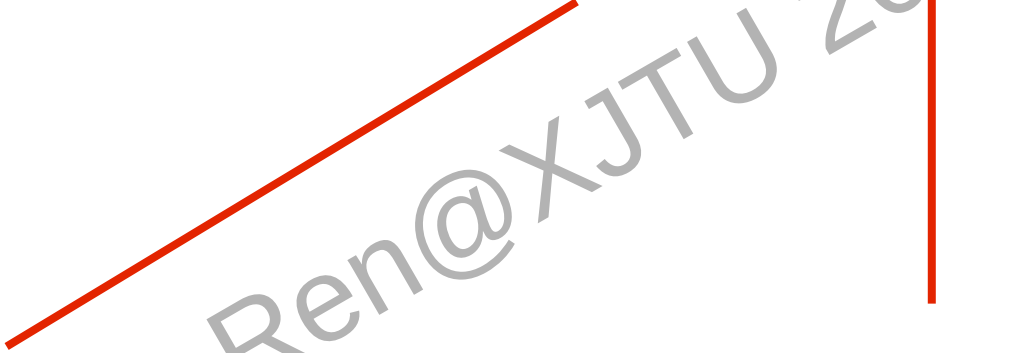
MediaTek MT8516

4 ARM Cortex-A35 CPU

NEON multimedia Processing Engine(SIMD)

What is a parallel computer

A parallel computer is a collection of processing elements that cooperate to solve problems quickly



We care about performance
***We care about efficiency**

We're going to use multiple processors to get it

One common definition of Parallel Computer

Speedup

One major motivation of using parallel processing: achieve a speedup

For a given problem:

$$\text{speedup(using P processors)} = \frac{\text{execution time (using 1 processor)}}{\text{execution time (using P processors)}}$$

DEMO 1

Class observations from demo 1

- **Observation: Speedup achieved, but communication limited the desired acceleration**
  - **In the demo, the communication was telling each other the partial sums**

- **Solution: minimizing the cost of communication improves speedup**
  - **Moving students ("processors") closer together (or let them shout)**

DEMO 2

(scaling up to four “processors”)

Class observations from demo 2

- **Observation:** imbalance in work assignment limited speedup
  - Some students ("processors") ran out work to do (went idle), while others were still working on their assigned task
- **Solution:** improving the **distribution of work** improved speedup

DEMO 3

(massively parallel execution)

Class observations from demo 3

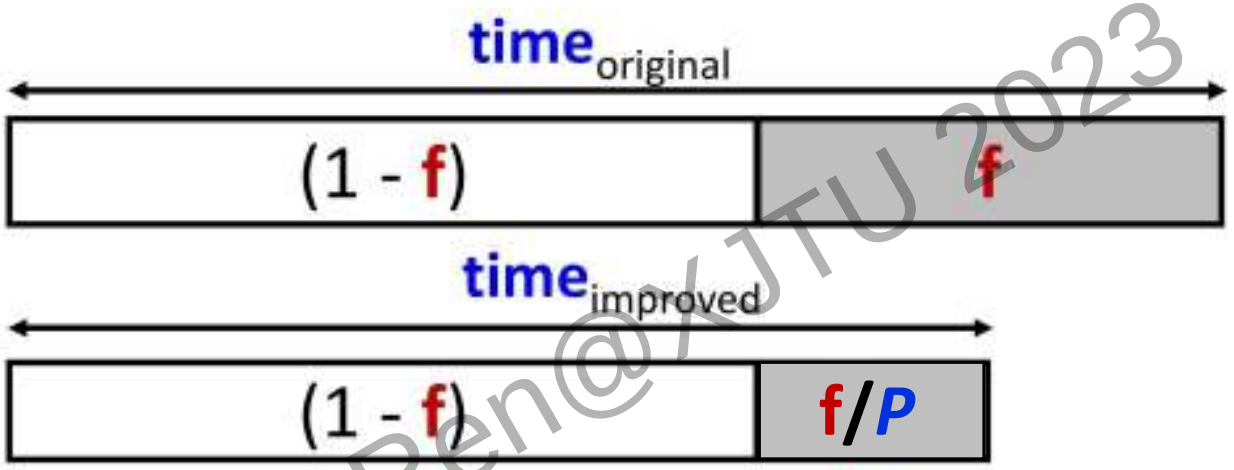
- **Observation:** The problem I just gave you has a significant amount of memory access latency compared to computation
- Bad memory access costs can dominate a parallel computation, severely limiting speedup

Ideal

Reality

Amdahl's Law on Speedup

If only a fraction *f* (of time) is speedup by *P*



$$\text{Speedup} = \frac{1}{(1-f)+f/P}$$

if f is small, *P* doesn't matter

even f is large, diminishing return on *P*, eventually "1-f" dominates

Last but not least : Amdahl's Law

Amdahl's Law states that potential program speedup is defined by the fraction of code (P) that can be parallelized

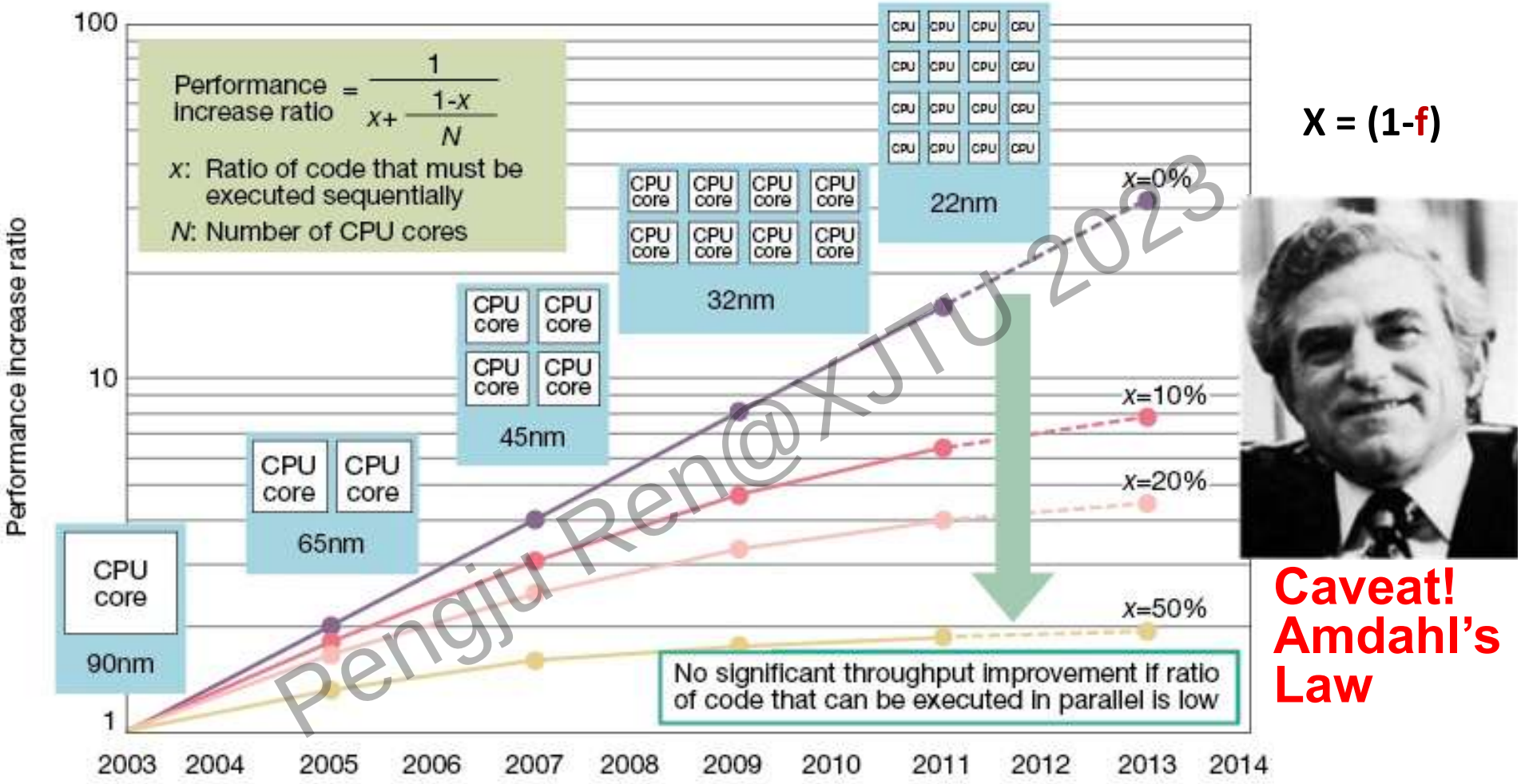
$$Speedup = \frac{1}{(1-f)+f/N}$$

It soon becomes obvious that there are limits to the scalability of parallelism:

	speedup			
N	P = .50	P = .90	P = .95	P = .99
10	1.82	5.26	6.89	9.17
100	1.98	9.17	16.80	50.25
1,000	1.99	9.91	19.62	90.99
10,000	1.99	9.91	19.96	99.02
100,000	1.99	9.99	19.99	99.90

Insufficient parallelism and **long-latency remote communication** are the two biggest performance challenges in using multiprocessors.

Challenges of Parallel Processing



X = (1-f)

Caveat! Amdahl's Law

Fig 3 Amdahl's Law an Obstacle to Improved Performance Performance will not rise in the same proportion as the increase in CPU cores. Performance gains are limited by the ratio of software processing that must be executed sequentially. Amdahl's Law is a major obstacle in boosting multicore microprocessor performance. Diagram assumes no overhead in parallel processing. Years shown for design rules based on Intel planned and actual technology. Core count assumed to double for each rule generation.

Theme1：Designing and writing parallel programs ... that scale

- **Parallel thinking**
  1. **Decomposing work into pieces that can safely be performed in parallel**
  2. **Assigning work to processors (may according to their specialty)**
  3. **Managing communication/synchronization between the processors so that it does not limit speedup**

- **Abstractions/mechanisms for performing the above tasks**
  - **Writing code in popular parallel programming languages**

Theme2：Parallel Computer hardware implementation: how parallel computers work

- **Mechanisms used to implement abstractions efficiently**
  - **Performance characteristics of implementations**
  - **Design trade-offs: performance vs. convenience vs. cost**

- **Why do I need to know about hardware?**
  - **Because the characteristics of the machine really matter**

  (recall speed of communication issues in earlier demos)
  - **Because you care about efficiency and performance ,**
    want to be the rare power programmer

  (you are writing parallel programs after all!)

Theme3：Thinking about efficiency

- **FAST != EFFICIENT**
- **Just because your program runs faster on a parallel computer, it does not mean it is using the hardware efficiently**
  - **Is 2x speedup on computer with 10 processors a good result?**
- **Programmer's perspective: make use of provided machine capabilities**
- **HW designer's perspective: choosing the right capabilities to put in system (performance/cost, cost = silicon area?, power?, etc.)**

Fundamental Shift in Processor Design Philosophy

Before 2004:

- within the chip area budget, maximize performance
  - » increasingly aggressive speculative execution(branch prediction) for ILP
  - » Increasing Caches

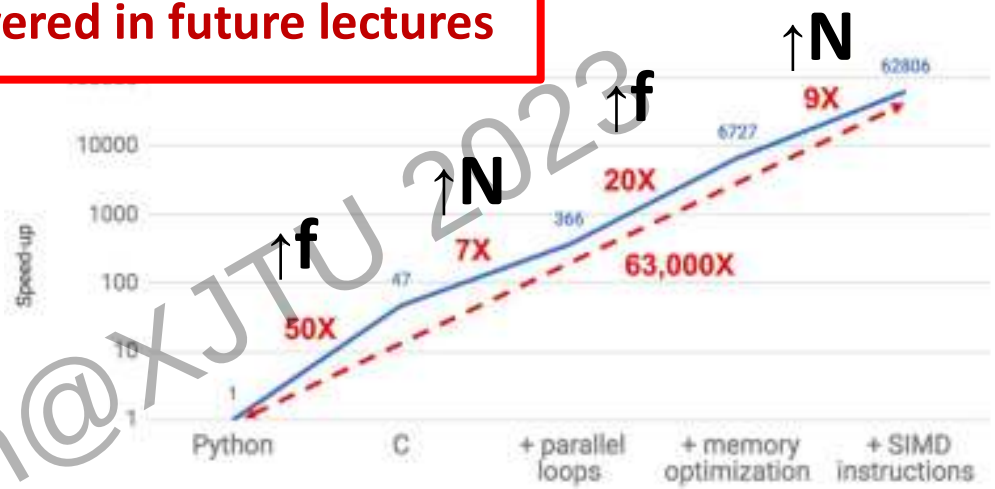
After 2004:

- area *within* the chip matters (limits # of cores/chip):
  - » maximize performance per area (PPA)
- power consumption is critical (battery life, data centers and IoT devices)
  - » maximize performance per Watt
- upshot: major focus on *efficiency* of cores

There is Plenty of ROOM at the Top (Above hardware)

Will be covered in future lectures

```
# 4096x4096 Matrix-multiplication:
for i in xrange(4096):
    for j in xrange(4096):
        for k in xrange(4096):
            C[i][j]+=A[i][k]*B[k][j]
```

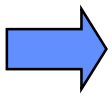


Version	Implementation	Running time (s)	GFLOPS	Absolute speedup	Relative speedup
1	Python	25,552.48	0.005	1	–
2	Java	2,372.68	0.058	11	10.8
3	C	542.67	0.253	47	4.4
4	Parallel loops	69.80	1.969	366	7.8
5	Parallel divide and conquer	3.80	36.180	6,727	18.4
6	plus vectorization	1.10	124.914	23,224	3.5
7	plus AVX intrinsics	0.41	337.812	62,806	2.7

There's plenty of room at the Top: What will drive computer performance after Moore's law? Leiserson, et. al., Science, 2020

Principles of Parallel Computing

- Finding enough parallelism (Amdahl's Law)
- Granularity – how big should each parallel task be
- Locality – moving data costs more than arithmetic
- Load balance – don't want 1K processors to wait for one slow one
- Coordination and synchronization – sharing data safely
- Performance modeling/debugging/tuning



All of these things makes parallel programming even harder than sequential programming.

Summary

- Today, single-thread/core performance is improving very slowly
  - To run programs significantly faster, programs must utilize multiple processing elements
  - Which means you need to know how to write parallel code
- Writing parallel programs can be challenging
  - Requires problem partitioning, communication, synchronization
  - Knowledge of machine characteristics is important

Next Lecture：Understanding Modern Processor：ILP and Optimization Code