Computer Architecture (Fall 2022)

Quantitative Approach

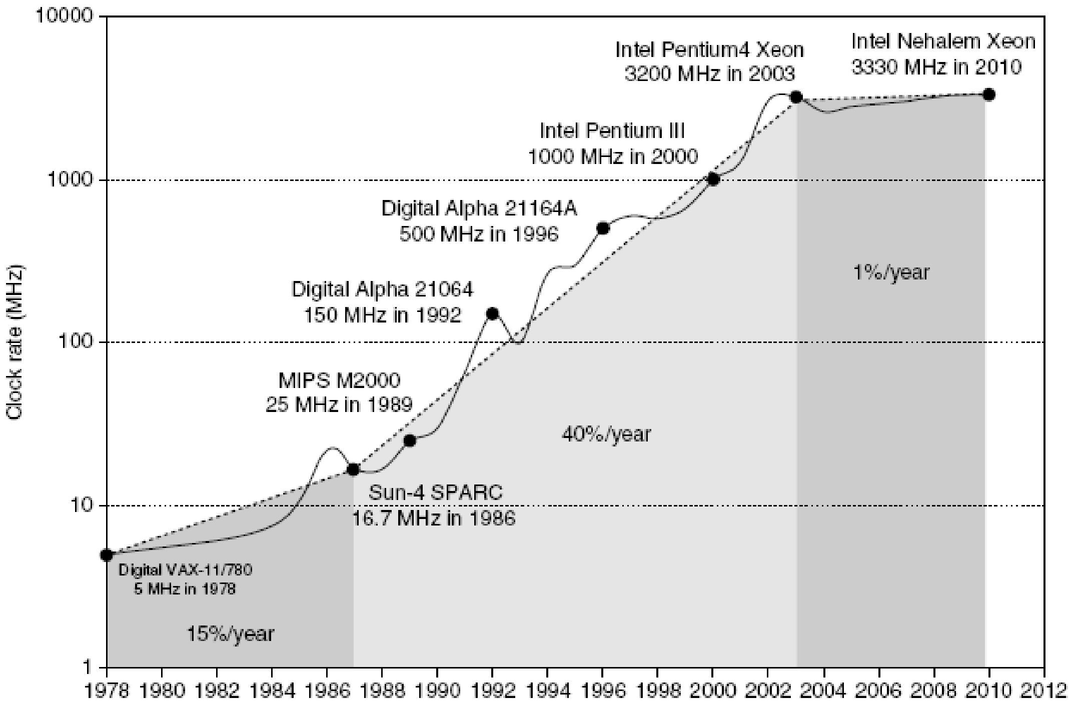
Dr. Duo Liu (刘铎)

Office: Main Building 0626

Email: liuduo@cqu.edu.cn

Power

- Intel 80386 consumed ~ 2 W
- 3.3 GHz Intel Core i7 consumes 130 W
- Heat must be dissipated from 1.5 x 1.5 cm chip
- This is the limit of what can be cooled by air



Dynamic Power Definition

- Dynamic energy
  - Transistor switch from 0 -> 1 or 1 -> 0

$$\text{Energy}_{\text{dynamic}} \propto 1/2 \times \text{Capacitive load} \times \text{Voltage}^2$$

- Dynamic power

$$\text{Power}_{\text{dynamic}} \propto 1/2 \times \text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency switched}$$

- Reducing clock rate reduces power, not energy

Reducing Dynamic Power

- Techniques for reducing power:
  - Do nothing well
  - Dynamic Voltage-Frequency Scaling (DVFS)
  - Low power state for DRAM, disks
  - Overclocking, turning off cores

Dynamic Voltage Frequency Scaling

- Example: H&P5th P23
  - DVFS is a low-power design technique that is becoming pervasive in modern processors
  - Example:

    If the voltage and frequency of a processing core are both reduced by 15% what would be the impact on dynamic power?

$$\text{Power Save} = \frac{P_{new}}{P_{old}} = \frac{C \times (V \times 0.85)^2 \times (F \times 0.85)}{C \times V^2 \times F} = 0.85^3 = 0.61$$

P_{new} is 61% more power efficient than P_{old}

Static Power Definition

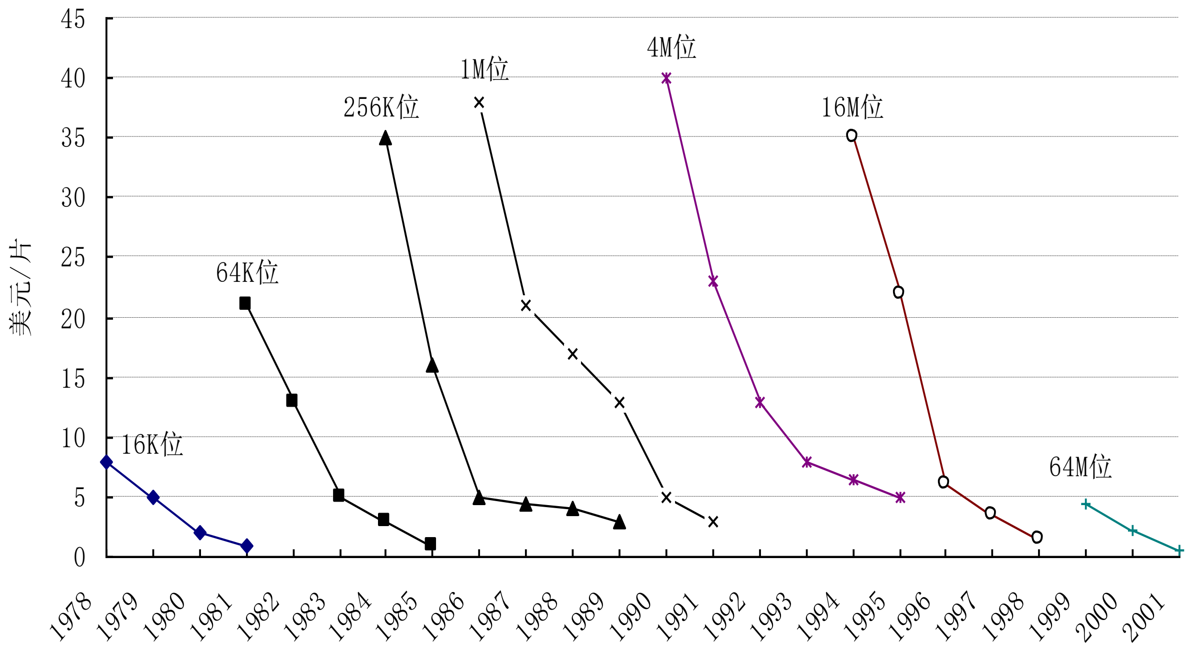
- Static power consumption

$$\text{Power}_{\text{static}} \propto \text{Current}_{\text{static}} \times \text{Voltage}$$

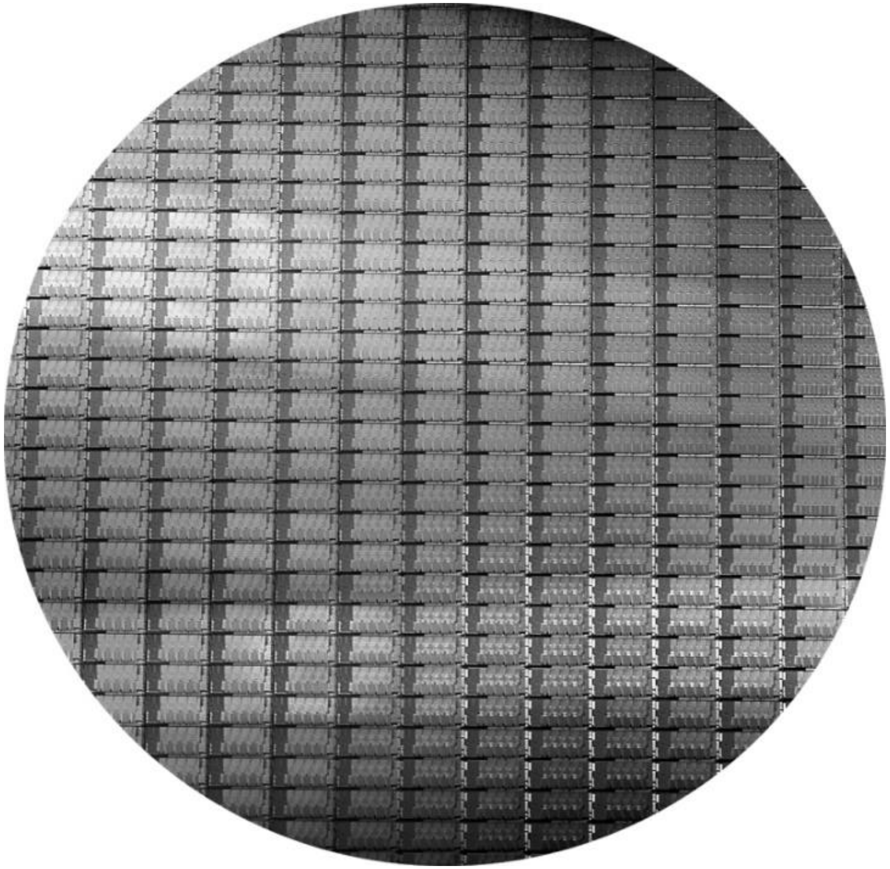
  - Scales with number of transistors
  - To reduce: power gating

Cost Trend

- **Cost driven down by learning curve**
  - Yield
- **DRAM: price closely tracks cost**
- **Microprocessors: price depends on volume**
  - 10% less for each doubling of volume



Integrated Circuit Cost



This 300 mm wafer contains 280 full Sandy Bridge dies, each 20.7 by 10.5 mm in a 32 nm process. (Sandy Bridge is Intel's successor to Nehalem used in the Core i7.) At 216 mm2, the formula for dies per wafer estimates 282. (Courtesy Intel.)

Integrated Circuit Cost

- **Integrated circuit**

$$\text{Cost of integrated circuit} = \frac{\text{Cost of die} + \text{Cost of testing die} + \text{Cost of packaging and final test}}{\text{Final test yield}}$$

$$\text{Cost of die} = \frac{\text{Cost of wafer}}{\text{Dies per wafer} \times \text{Die yield}}$$

$$\text{Dies per wafer} = \frac{\pi \times (\text{Wafer diameter}/2)^2}{\text{Die area}} - \frac{\pi \times \text{Wafer diameter}}{\sqrt{2 \times \text{Die area}}}$$

- **Bose-Einstein formula:**

$$\text{Die yield} = \text{Wafer yield} \times 1/(1 + \text{Defects per unit area} \times \text{Die area})^N$$

- **Defects per unit area = 0.016-0.057 defects per square cm(2010)**
- **N = process-complexity factor = 11.5-15.5 (40 nm, 2010)**

Integrated Circuit Cost: Example

- **Example: H&P 5th P31**

Find the number of dies per 300 mm (30 cm) wafer for a die that is 1.5 cm on a side and for a die that is 1.0 cm on a side.

- **Answer**

When die area is 2.25 cm^2:

$$\text{Dies per wafer} = \frac{\pi \times (30/2)^2}{2.25} - \frac{\pi \times 30}{\sqrt{2 \times 2.25}} = \frac{706.9}{2.25} - \frac{94.2}{2.12} = 270$$

Since the area of the larger die is 2.25 times bigger, there are roughly 2.25 as many smaller dies per wafer:

$$\text{Dies per wafer} = \frac{\pi \times (30/2)^2}{1.00} - \frac{\pi \times 30}{\sqrt{2 \times 1.00}} = \frac{706.9}{1.00} - \frac{94.2}{1.41} = 640$$

Integrated Circuit Cost: Example

- **Example: H&P 5th P31**

Find the die yield for dies that are 1.5 cm on a side and 1.0 cm on a side, assuming a defect density of 0.031 per cm^2 and N is 13.5.

- **Answer**

The total die areas are 2.25 cm^2 and 1.00 cm^2. For the larger die, the yield is

$$\text{Die yield} = 1/(1 + 0.031 \times 2.25)^{13.5} = 0.40$$

For the smaller die, the yield is

$$\text{Die yield} = 1/(1 + 0.031 \times 1.00)^{13.5} = 0.66$$

That is, less than half of all the large dies are good but two-thirds of the small dies are good.

How to Define Performance?

Airplane	Passenger Capacity	Cruising Range (miles)	Cruising Speed (m.p.h.)	Passenger Throughput (passenger x m.p.h)
Boeing 777	370	4630	610	225,700
Boeing 747	470	4150	610	286,700
Concorde	132	4000	1350	178,200
Douglas DC-8-50	146	8720	544	79,424

Two Key Performance Metrics

- Time to run the task
  - execution time, response time, elapsed time, latency
- Tasks per time unit
  - execution rate, bandwidth, throughput

Airplane	DC to paris	Speed	Passengers	Throughput (passengers x mph)
Boeing 747	6.5 hours	610 mph	470	286,700
Concorde	3 hours	1350 mph	132	178,200

Latency vs. Throughput

- Latency
  - “real” time necessary to complete a task
  - important when the focus is on a single task
    - a computer user who is working with a single application
    - a critical task of a real-time embedded system
- Throughput (aka Bandwidth)
  - number of tasks completed per unit of time
  - a metric independent from the exact number of executed tasks
  - important when the focus is on running many tasks
    - a manager of a large data-processing center is interested in the total amount of work done in a given time

Example: the Classic 5-Stage Pipeline

Instr. No.	Pipeline Stage						
1	IF	ID	EX	MEM	WB		
2		IF	ID	EX	MEM	WB	
3			IF	ID	EX	MEM	WB
4				IF	ID	EX	MEM
5					IF	ID	EX
Clock Cycle	1	2	3	4	5	6	7

- Pipelining
  - increases the instruction throughput
    - number of instructions completed per unit of time
  - but does not reduce (in fact, it usually slightly increases) the execution time of an individual instruction

Performance Metrics

- Machine X is n times faster than machine Y

$$n = \frac{executionTime(Y)}{executionTime(X)} = \frac{performance(X)}{performance(Y)}$$

- Performance and execution time are reciprocal
  - improve performance → increase performance
  - improve execution time → decrease execution time
- Example
  - executionTime(Y) = 4.8, executionTime(X) = 3.6
    - n = 1.33, i.e. X is 33% faster than Y

Benchmark Suites

- Sets of programs to simulate typical workloads
- Several types
  - real software applications (GCC, Word,…)
    - most accurate but typically longer to process
    - portability problems (OS/compiler dependencies), GUI
  - kernels (Livermore Loops, Linpack,…)
    - small, key pieces taken from real programs
    - limited picture, but good to isolate the performance of individual features of a machine
  - synthetic benchmarks (Whetstone, Dhrystone,…)
    - try to match the average frequency of operations on operands of a real program
      - may easily mislead compiler and hardware designers

Principle of Locality

- Temporal Locality
  - a resource that is referenced at one point in time will be referenced again sometime in the near future
- Spatial Locality
  - the likelihood of referencing a resource is higher if a resource near it was just referenced
- 90/10 Locality Rule of Thumb
  - a program spends 90% of its execution time in only 10% of its code
    - hence, it is possible to predict with reasonable accuracy what instructions and data a program will use in the near future based on its accesses in the recent past
    - **this is a consequence of how we program and we store the data in the memory**

Principle of Locality - Example

- Cache
  - directly exploits temporal locality providing faster access to a smaller subset of the main memory which contains copy of data recently used
  - but, all data in the cache are not necessarily data that are spatially close in the main memory…
  - …still, when a cache miss occurs a fixed-size block of contiguous memory cells is retrieved from the main memory based on the principle of spatial locality