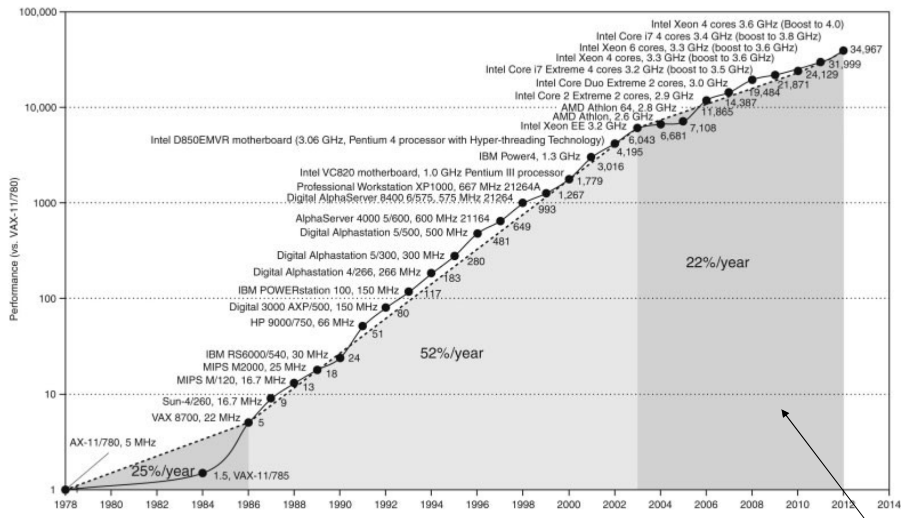
Introduction

Why study Hardware?

- ✓ Decline of Moore’s Law (#transistors per silicon chip doubles every 18-24 months)
- ✓ Proliferation of multi-core processors
- ✓ Emergence of new platforms (e.g., cell phones, automobiles)

Hardware knowledge helps programmers (chip/OS/compiler) to write better code

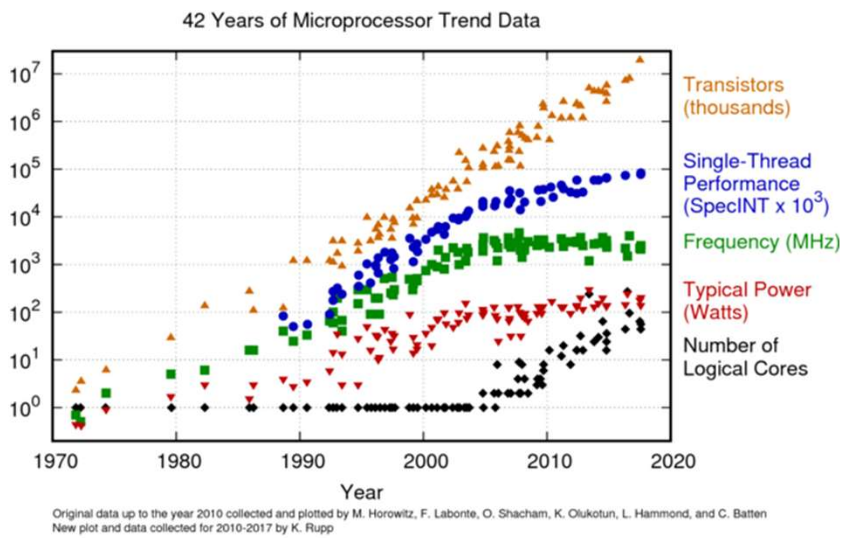
Uniprocessor Performance?



50% improvement every year!
What contributes to this improvement?

Constrained by power wall

Microprocessor Performance



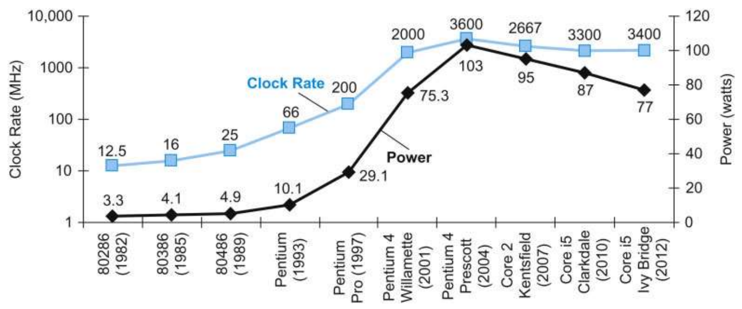
Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2017 by K. Rupp

Power Consumption Trends

- ✓ Dynamic power ∝ activity × capacitance × voltage^2 × frequecy

Q. What is the effect of Moore's Law of scaling on the dynamic power equation?

- ✓ Voltage and frequency are somewhat constant now, while capacitance per transistor is decreasing and number of transistors (activity) is increasing
- ✓ Leakage power is also rising (function of #transistor and voltage)



Summary and Important Trends

Summary:

- ✓ Increasing frequency led to power wall in early 2000s
- ✓ Frequency has stagnated since then
- ✓ End of voltage scaling in early 2010s

Trends:

- ✓ Running out of ideas to improve single thread performance
- ✓ Power wall makes it harder to add complex features
- ✓ Power wall makes it harder to increase frequency
- ✓ Additional performance provided by: more cores, occasional spikes in frequency, accelerators

Important Trends

Historical contributions to performance:
1. Better processes (faster devices) ~20%
2. Better circuits/pipelines ~ 15%
3. Better organization/architecture ~ 15%

In the future, (2) will help little and (1) will eventually disappear!

	Pentium	P-Pro	P-II	P-III	P-4	Itanium	Montecito
Year	1993	1995	1997	1999	2000	2002	2005
Transistor	3.1M	5.5M	7.5M	9.5M	42M	300M	1720M
Clock speed	60M	200M	300M	500m	1500M	800M	1800M

$\uparrow$

Moore's Law in action

At this point, adding transistors
to a core yields little benefit

What Does This Mean to a Programmer

Today, one can expect only a 20% annual improvement; the improvement is even lower if the program is not multi-threaded

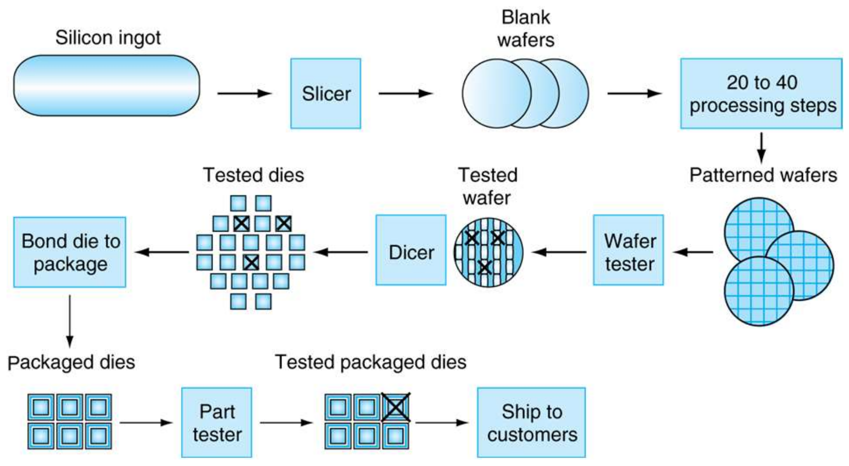
- ✓ A program needs many threads
- ✓ The threads need efficient synchronization and communication
- ✓ Data placement in the memory hierarchy is important
- ✓ Accelerators should be used when possible

Challenges for Hardware Designers

Find efficient ways to

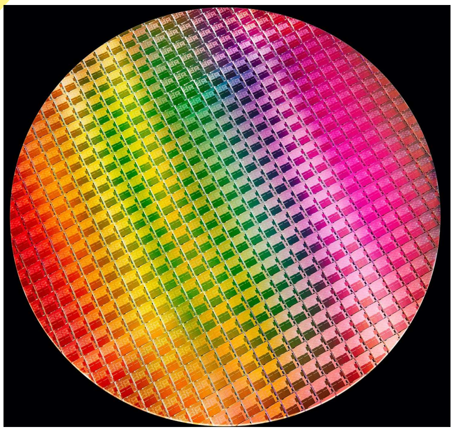
- ✓ improve single-thread performance and energy
- ✓ improve data sharing
- ✓ boost programmer productivity
- ✓ manage the memory system
- ✓ build accelerators for important kernels
- ✓ provide security

Manufacturing ICs



Yield: proportion of working dies per wafer

Intel® Core 10th Gen



300mm wafer, 506 chips, 10nm technology

Each chip is 11.4 x 10.7 mm

$$Cost\ per\ die = \frac{Cost\ per\ wafer}{Dies\ per\ wafer\ \times Yield}$$

$$Dies\ per\ wafer\ \approx \frac{Wafer\ area}{Die\ area}$$

$$Yield = \frac{1}{(1 + \frac{Defects\ per\ area \times Die\ area}{2})^2}$$

- Nonlinear relation to area and defect rate
  - Wafer cost and area are fixed
  - Defect rate determined by manufacturing process
  - Die area determined by architecture and circuit design

Processor Technology Trends

✓ Shrinking of transistor sizes: 250nm (1997) → 130nm (2002) → 70nm (2008) → 35nm (2014) → 10nm (2019) → now transitioning to 7nm

✓ Transistor density increases by 35% per year and die size increases by 10-20% per year… functionality improvements!

✓ Transistor speed improves linearly with size (complex equation involving voltages, resistances, capacitances)

✓ Wire delays do not scale down at the same rate as transistor delays

Memory and IO Technology Trends

- ✓ DRAM density increases by 40-60% per year, latency has reduced by 33% in 10 years (the memory wall!), bandwidth improves twice as fast as latency decreases
- ✓ Disk density improves by 100% every year, latency improvement similar to DRAM
- ✓ Networks: primary focus on bandwidth; 10Mb → 100Mb in 10 years; 100Mb → 1Gb in 5 years

The HW/SW Interface

Application software

Systems software (OS, compiler)

Hardware

```
a[i] = b[i] + c;
        | Compiler
        v
lw    $15, 0($2)
add   $16, $15, $14
add   $17, $15, $13
lw    $18, 0($12)
lw    $19, 0($17)
add   $20, $18, $19
sw    $20, 0($16)
        | Assembler
        v
000000101100000
110100000100010
...
```

Performance Metrics

- Possible measures:
  - response time – time elapsed between start and end of a program
  - throughput – amount of work done in a fixed time
- How are response time and throughput affected by
  Replacing the processor with a faster version?
  Adding more processors?
- **Note:** we will be primarily concerned with response time

Relative Performance

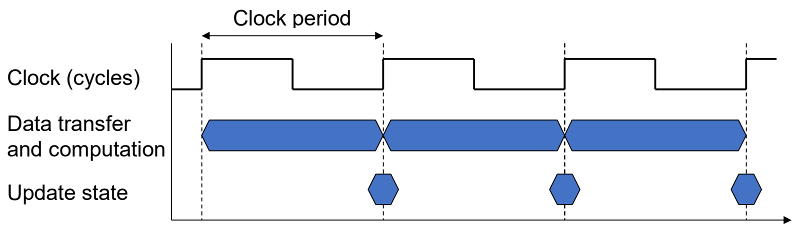
- Define Performance = 1/Execution Time
- "X is *n* time faster than Y"

$$\frac{Performance_X}{Performance_Y} = \frac{Execution\ time_Y}{Execution\ time_X} = n$$

- **Example**: time taken to run a program
  - 10s on A, 15s on B
  - Execution Time$_B$ / Execution Time$_A$
    = 15s / 10s = 1.5
  - So A is 1.5 times faster than B

CPU Clocking

- Operation of digital hardware governed by a constant-rate clock



- Clock period: duration of a clock cycle
  - e.g., 250ps = 0.25ns = 250×10^{-12}s
- Clock frequency (rate): cycles per second
  - e.g., 4.0GHz = 4000MHz = 4.0×10^{9}Hz

CPU Time

$$\text{CPU execution time for a program} = \text{CPU clock cycles for a program} \times \text{Clock cycle time}$$

$$= \text{CPU clock cycles for a program} \;/\; \text{Clock rate}$$

Performance improved by

- Reducing number of clock cycles
- Increasing clock rate
- Hardware designer must often trade off clock rate against cycle count

CPU Time

$$\text{CPU execution time for a program} = \text{CPU clock cycles for a program} \times \text{Clock cycle time} = \text{CPU clock cycles for a program} \,/\, \text{Clock rate}$$

Example: A program runs on

- Computer A: 2GHz clock, 10s CPU time
- Designing computer B:
  - Aim for 6s CPU time
  - Can have faster clock, but the faster clock affect the rest of CPU design, causing machine B to require 1.2 times as many clock cycles as machine A to execute the program
- How fast must Computer B clock be?

Instruction Count (IC) and Cycles per Instruction (CPI)

Clock cycles = Instruction count × Cycles per instruction

CPU time = Instruction count × Cycles per instruction × Clock cycle time

- Instruction Count for a program
  - Determined by program, ISA and compiler
- Average cycles per instruction
  - Determined by CPU hardware
  - If different instructions have different CPI: Average CPI affected by instruction mix

Example

Clock cycles = Instruction count × Cycles per instruction

CPU time = Instruction count × Cycles per instruction × Clock cycle time

Which of the following two systems is better?

1. A program is converted into 4 billion MIPS instructions by a compiler ; the MIPS processor is implemented such that each instruction completes in an average of 1.5 cycles and the clock speed is 1 GHz

2. The same program is converted into 2 billion x86 instructions; the x86 processor is implemented such that each instruction completes in an average of 6 cycles and the clock speed is 1.5 GHz

CPI for Different Instruction Classes

- If different instruction classes take different numbers of cycles

Clock cycles = $\sum_{i=1}^{n}(CPI_i \times IC_i)$

- Weighted average CPI

$$CPI = \frac{Clock\ cycles}{IC} = \sum_{i=1}^{n}\left(CPI_i \times \underbrace{\frac{IC_i}{IC}}\right)$$

Relative frequency

CPI Example

- Alternative compiled code sequences using instructions in classes A, B, C

Class	A	B	C
CPI for class	1	2	3
IC in sequence 1	2	1	2
IC in sequence 2	4	1	1

- Sequence 1: IC = 5
  - Clock Cycles
    = 2×1 + 1×2 + 2×3
    = 10
  - Avg. CPI = 10/5 = 2.0
- Sequence 2: IC = 6
  - Clock Cycles
    = 4×1 + 1×2 + 1×3
    = 9
  - Avg. CPI = 9/6 = 1.5

Performance Summary

The BIG Picture

$$\text{CPU Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$$

- **Performance depends on**
  - Algorithm: affects IC, possibly CPI
  - Programming language: affects IC, CPI
  - Compiler: affects IC, CPI
  - Instruction set architecture: affects IC, CPI, T_c

Power and Energy

- Total power = dynamic power + leakage power
- Dynamic power ∝ activity × capacitance × voltage^2 × frequecy
- Leakage power ∝ voltage
- Energy (J) = power (w) × time (sec.)

Example: A 1 GHz processor takes 100 seconds to execute a program, while consuming 70 W of dynamic power and 30 W of leakage power. Does the program consume less energy in Turbo boost mode when the frequency is increased to 1.2 GHz?

Normal mode energy = 100 W × 100 s = 10000 J
Turbo mode energy = (70 × 1.2 + 30) × 100/1.2 = 9500 J
Note: Frequency only impacts dynamic power, not leakage power. We assume that the program's CPI is unchanged when frequency is changed, i.e., execution time varies linearly with cycle time

Amdahl's Law

Amdahl's Law: performance improvements through an enhancement is limited by the fraction of time the enhancement comes into play

Example: Suppose a programs runs in 100 sec on a machine, with multiply operations responsible for 80 sec of this time. How much do you have to improve the speed of multiplication if you want my program to run five times faster?

$$\begin{array}{c}\textit{Execution time}\\ \textit{after improvement}\end{array} = \left(\frac{\begin{array}{c}\textit{Execution time affected}\\ \textit{by improvement}\end{array}}{\textit{Amount of improvement}}\right) + \textit{Execution time unaffected}$$

$$20 = \frac{80}{n} + 20$$

- Architecture design is very bottleneck-driven – make the common case fast, do not waste resources on a component that has little impact on overall performance/power

Conclusion

- ➢Cost/performance is improving
  - Due to underlying technology development
- ➢Hierarchical layers of abstraction
  - In both hardware and software
- ➢Instruction set architecture
  - The hardware/software interface
- ➢Execution time: the best performance measure
- ➢Power is a limiting factor
  - Use parallelism to improve performance