ECE-425: Lecture 2 Introduction to computer architecture

Prof: Mohamed El-Hadedy

Email: mealy@cpp.edu

Office: 909-869-2594

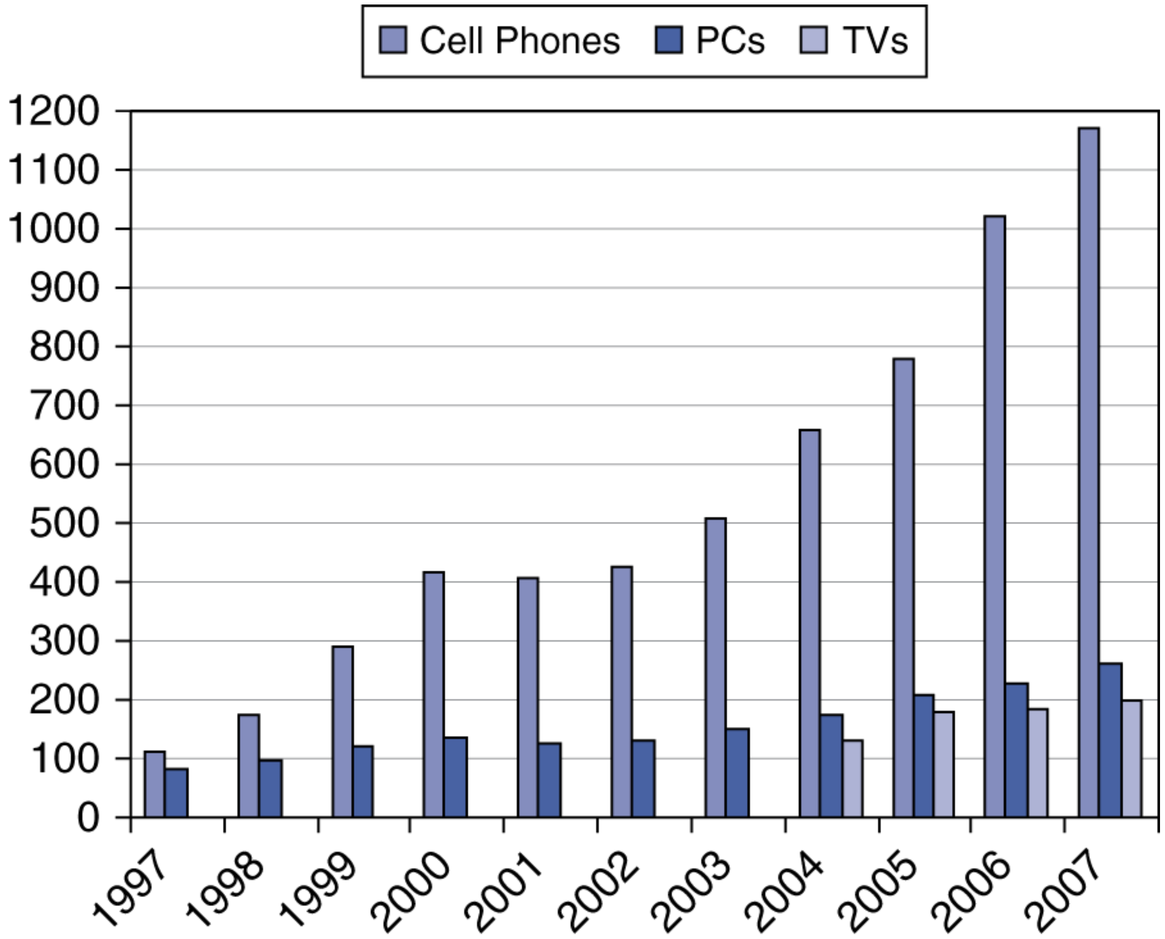
The Computer Revolution

- Progress in computer technology
- Makes novel applications feasible
  - Computers in automobiles
  - Cell phones
  - World Wide Web
  - Search Engines
- Computers are pervasive

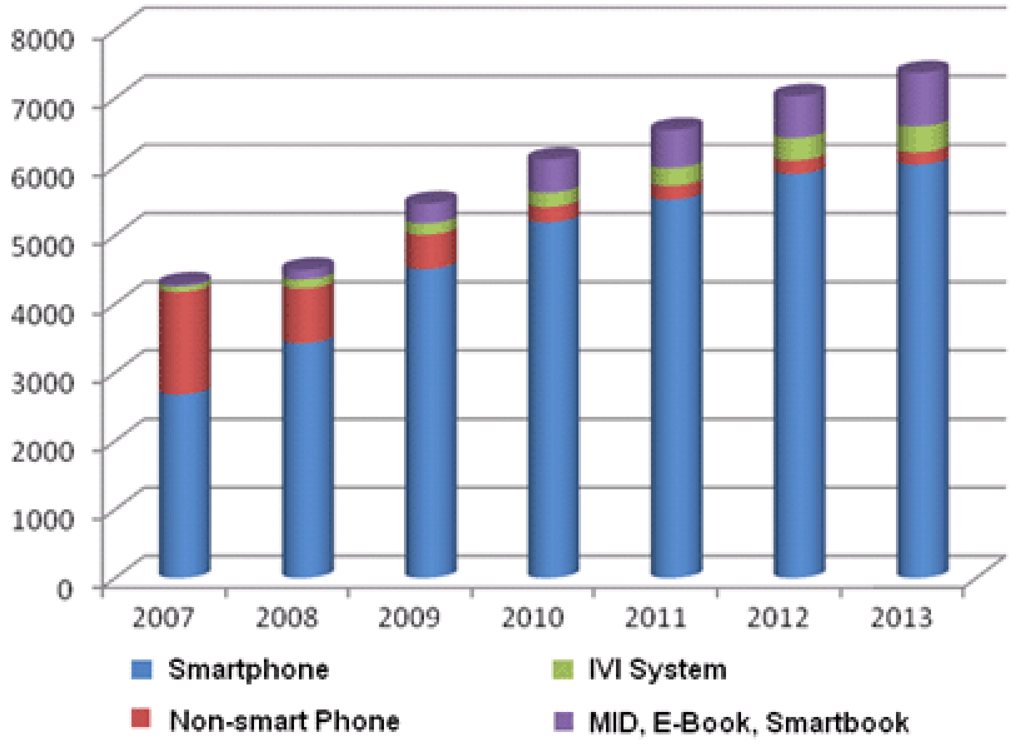
Classes of Computers

- **Desktop computers**
  - General purpose, variety of software
  - Subject to cost/performance tradeoff
- **Server computers**
  - Network based
  - High capacity, performance, reliability
  - Range from small servers to large sized
- **Embedded computers**
  - Hidden as components of systems
  - Power/performance/cost constraints

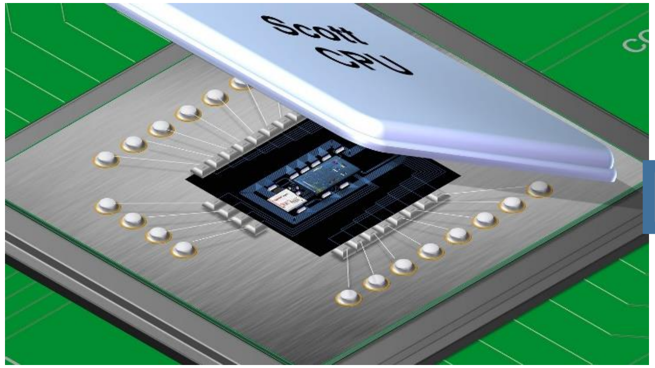
The Processor Market

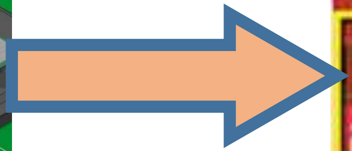


Smartphone Chip Makers

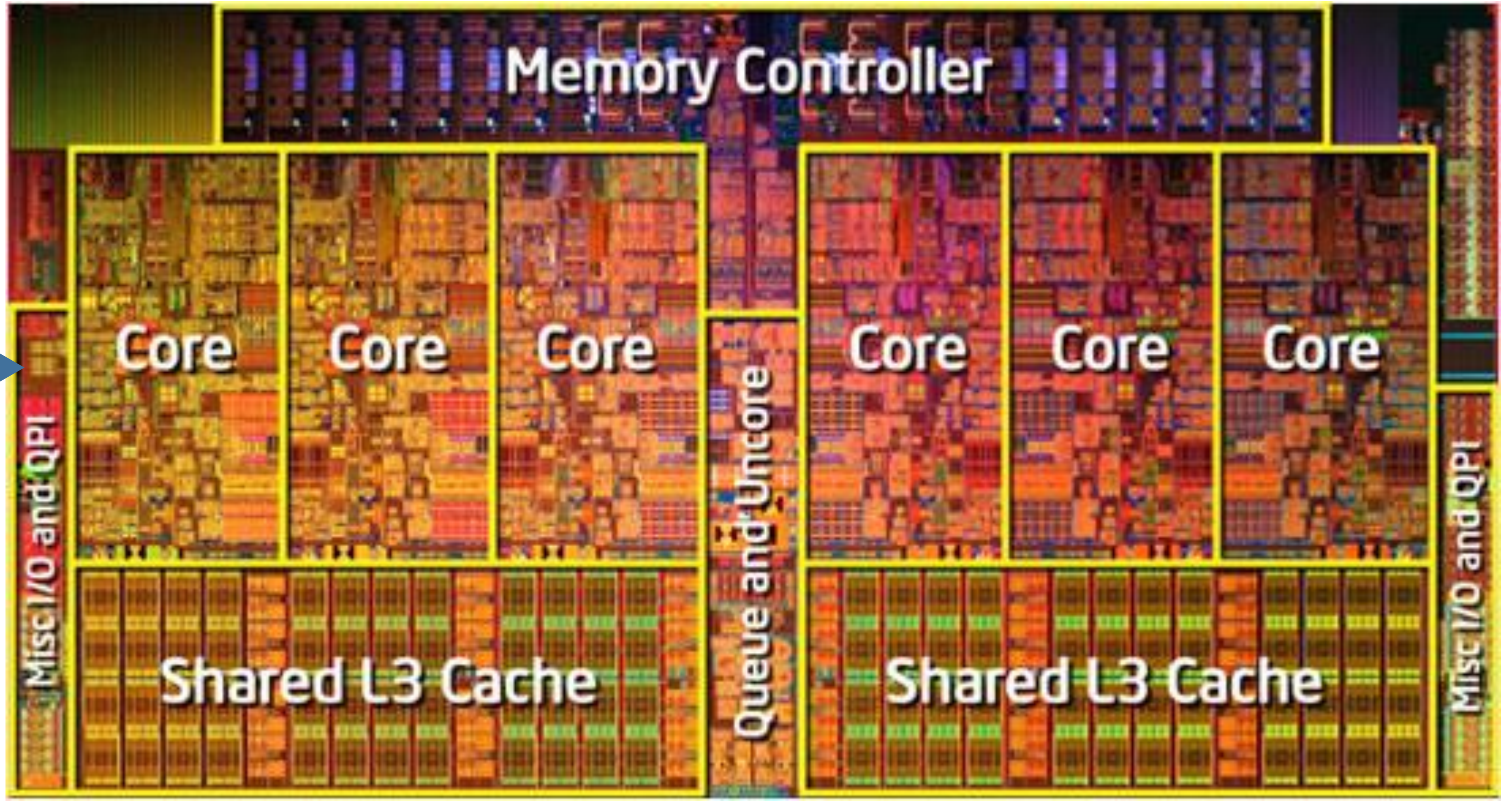


Inside the Processor



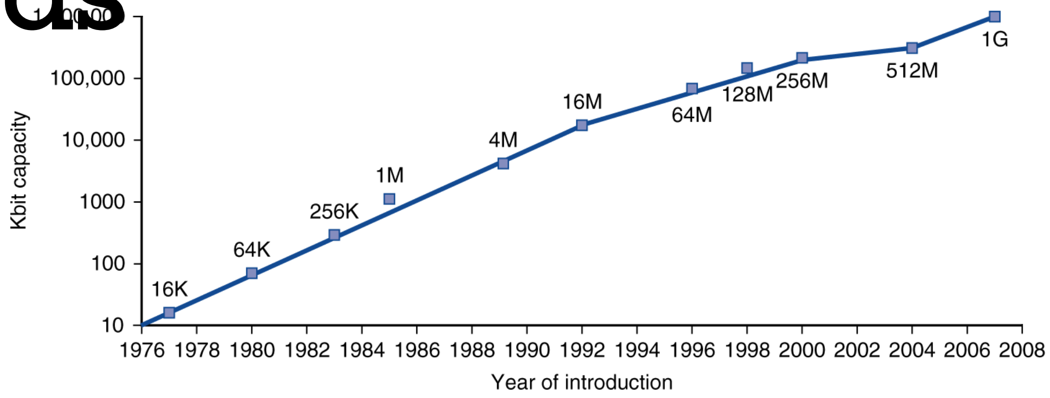


Intel Core i7-980X Six-Core Processor



Technology Trends

- Electronics technology continues to evolve
  - Increased capacity and performance
  - Reduced cost



DRAM capacity

Year	Technology	Relative performance/cost
1951	Vacuum tube	1
1965	Transistor	35
1975	Integrated circuit (IC)	900
1995	Very large scale IC (VLSI)	2,400,000
2005	Ultra large scale IC	6,200,000,000

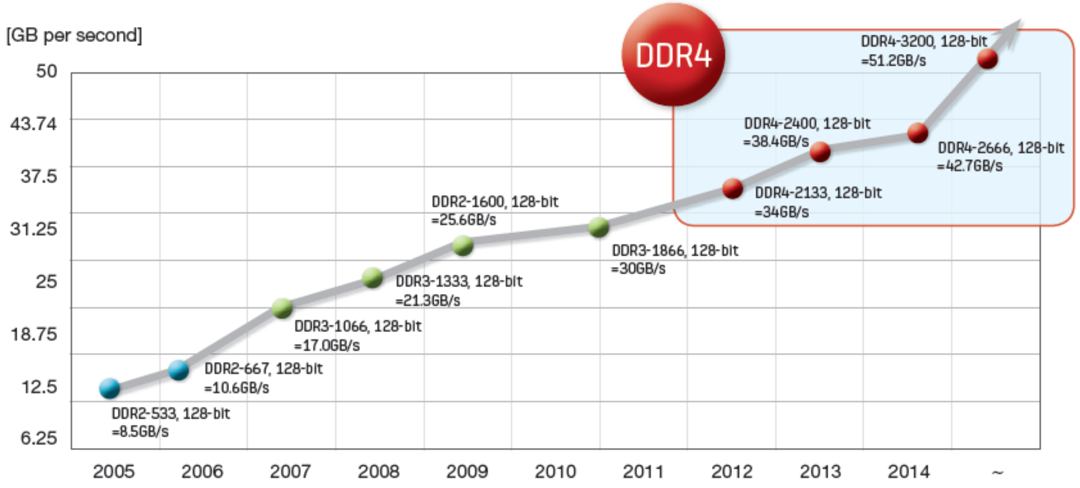


Figure 2. DDR4 higher performance compared with DDR3L and DDR2

Memory Performance

Performance metrics

- **Why study performance metrics?**
  - Determine the benefit/lack of benefits of designs
  - Computer design is too complex to intuit performance & performance bottlenecks
  - Have to be careful about what you mean to measure & how you measure it

- **What you should get out of this discussion**
  - Good metrics for measuring computer performance
  - What they should be used for
  - What metrics you shouldn’t use & how metrics are misused

Understanding Performance

- **Algorithm**
  - Determines number of operations executed
- **Programming language, compiler, architecture**
  - Determine number of machine instructions executed per operation
- **Processor and memory system**
  - Determine how fast instructions are executed
- **I/O system (including OS)**
  - Determines how fast I/O operations are executed

Performance of computer systems

Many different factors to take into account when determining performance:

- **Technology**
  - ❑ Circuit speed (clock, MHz)
  - ❑ Processor technology (how many transistors on a chip)
- **Organization**
  - ❑ Type of processor (ILP: Instruction-level parallelism)
  - ❑ Configuration of the memory hierarchy
  - ❑ Type of I/O devices
  - ❑ Number of processors in the system
- **Software**
  - ❑ Quality of the compilers
  - ❑ Organization & quality of OS (Operating system), databases, etc.

Performance of computer systems

Purchasing perspective

Which machine has the best performance ?

- least cost ?
- best performance / cost ?

Design perspective

Which design has the best performance improvement ?

- least cost ?
- best performance / cost ?

Metrics that Measure Performance

- **Execution time:** time to execute one program from beginning to end
- **Response time:** How long it takes to do a task
- **Throughput:** total amount of work completed in a given time
  - Transactions (Database) or packets (webservers)/second
  - An indication of how well hardware resources are being used
  - Good metrics for chip designers or managers of computer systems
  - **Note:** Often improving execution time will improve throughput & vice versa.)
- **Component metrics:** subsystem performance, e.g., memory behavior
  - Help explain how execution time was obtained
  - Pinpoints performance bottlenecks

Execution time & Relative Performance

- **Performance = (1/Execution time)**
- How are response time and throughput affected by
  - o Replacing the processor with a faster version?
  - o Adding more processors?
- Comparing between two processors A, B
  - Performance $_A$/ Performance $_B$ = n → "A is n times faster than B"
  - Execution time of A is n times longer than B

Example: time taken to run a program

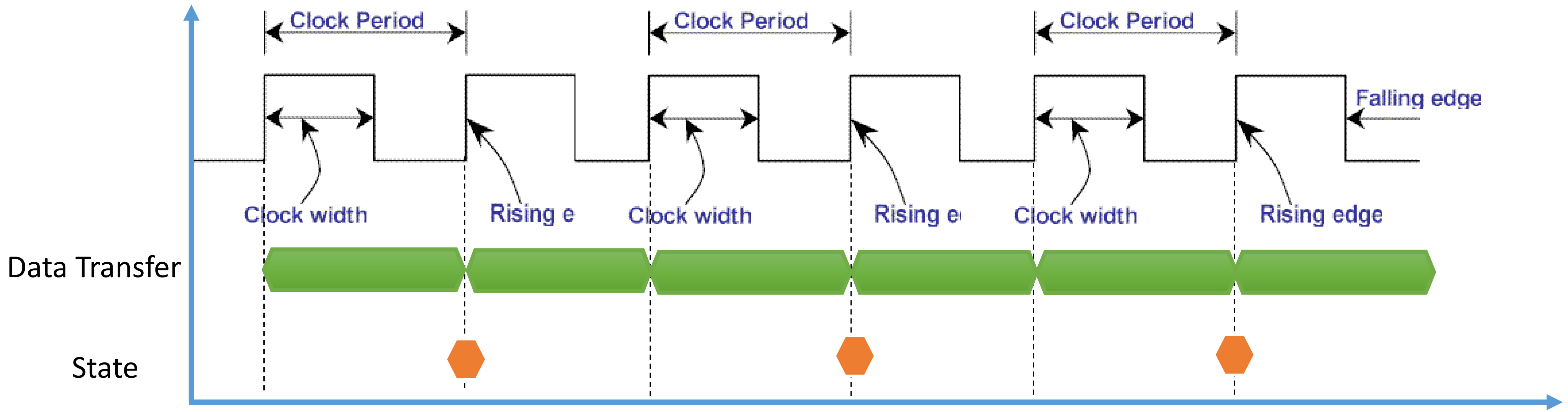
- 20s on A, 25s on B
- Execution time$_B$/ Execution time$_A$ = 25/20 = 1.25
- So A is 1.25 times faster than B

Measuring Execution Time

- **Elapsed time**
  - Elapsed time
    - ✓ Total response time, including all aspects
      - o Processing, I/O, OS overhead, idle time
    - ✓ Determines system performance
- ▪ CPU Execution Time: The time the CPU spends executing an application
  - ▪ No memory effects
  - ▪ No I/O time
  - ▪ No effects of multiprogramming
- ▪ CPUExecution Time: = CPUClockCycles * ClockCycle Time

CPU Execution Time (cont.)

- Cycle time (clock period) is measured in time or rate
  - ❑ Clock cycle time = 1/clock cycle rate
- ❑ CPUExecutionTime = CPUClockCycles/ClockCycleRate



- **Clock period:** Duration of a clock cycle
- **Clock Frequency (rate):** Cycles per second
- Clock cycle rate of 1 MHz = cycle time of 1 μS
- Clock cycle rate of 1 GHz = cycle time of 1 nS

CPU Execution Time (cont.)

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

CPU Execution Time Example

- **Computer A:** 2 GHz clock, 10s CPU time
- **Designing Computer B:**
  - Aim for 6S CPU time
  - Can do faster clock, but causes 1.2 x Clock cycles
- **How fast must Computer B clock be?**

Solution:

Clock Rate (B) = (Clock Cycles (B) / CPU Time (B)) = 1.2 x Clock Cycles (A) / 6S

Clock Cycles (A) = CPU Time (A) X Clock Rate (A) = 10s x 2 GHz = 20×10^9

Clock Rate (B) = $(1.2 \times 20 \times 10^9)/6S = 24 \times 10^9/6s$ = 4 GHz

Number of Cycles per Instruction (CPI)

- Different instructions needs different # of cycles:
  - Multiplications takes more time than addition (Multiplications is adding and shifting)
  - Floating point instructions take longer than integer instructions
  - Accessing memory takes more time than accessing registers
  - **Note:** Changing the cycle time often changes the number of cycles required for various instructions
- CPU Clock Cycles = Number Of Instructions X CPI
- CPU Time = Number Of Instructions X CPI X Clock Cycle Time = Number of Instructions X CPI / Clock Rate

Number of Cycles per Instruction (CPI)

- Average number of clock cycles per instruction
  - ✓ Throughput metric
  - ✓ Component metric, not a measure of performance
  - ✓ Used for processor organization studies, given a fixed compiler & ISA

- Average cycles per instruction
  - Determined by CPU hardware
  - If different instructions have different CPI
    - Average CPI affected by instruction mix

CPI Example

- Computer A: Cycle Time = 250ps, CPI = 2.0
- Computer B: Cycle Time = 500 ps, CPI = 1.2
- Same ISA architecture (Instruction count is the same I)
- Which faster, and by how much?

Solution

- CPU Time (A) = I x 2.0 x 250 ps = I x 500 ps
- CPU Time (B) = I x 1.2 x 500 ps = I x 1.2 x 500 ps = I x 600 ps
- CPU Time (B) / CPU Time (A) = I x 600 ps / I x 500 ps = 1.2
- A is faster than B by 1.2

CPI (Different classes of instructions)

- If different instruction classes take different number of cycles

$$CPUClockCycles = \sum_{i=1}^{n} (\mathrm{CPI}_i \times \mathrm{C}_i)$$

- Where CPI_i = CPI for a particular class of instructions
- where C_i = the number of instructions of i^{th} class that have been executed
- Improving part of the architecture can improve a CPI_i

Weighted average CPI

$$\mathrm{CPI} = \frac{\mathrm{CPUClockCycles}}{\text{Number of instructions}} = \sum_{i=1}^{n} \left(\mathrm{CPI}_i \times \underbrace{\frac{\mathrm{C}_i}{\text{Number of Instructions}}}_{\text{Relative frequency}} \right)$$

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

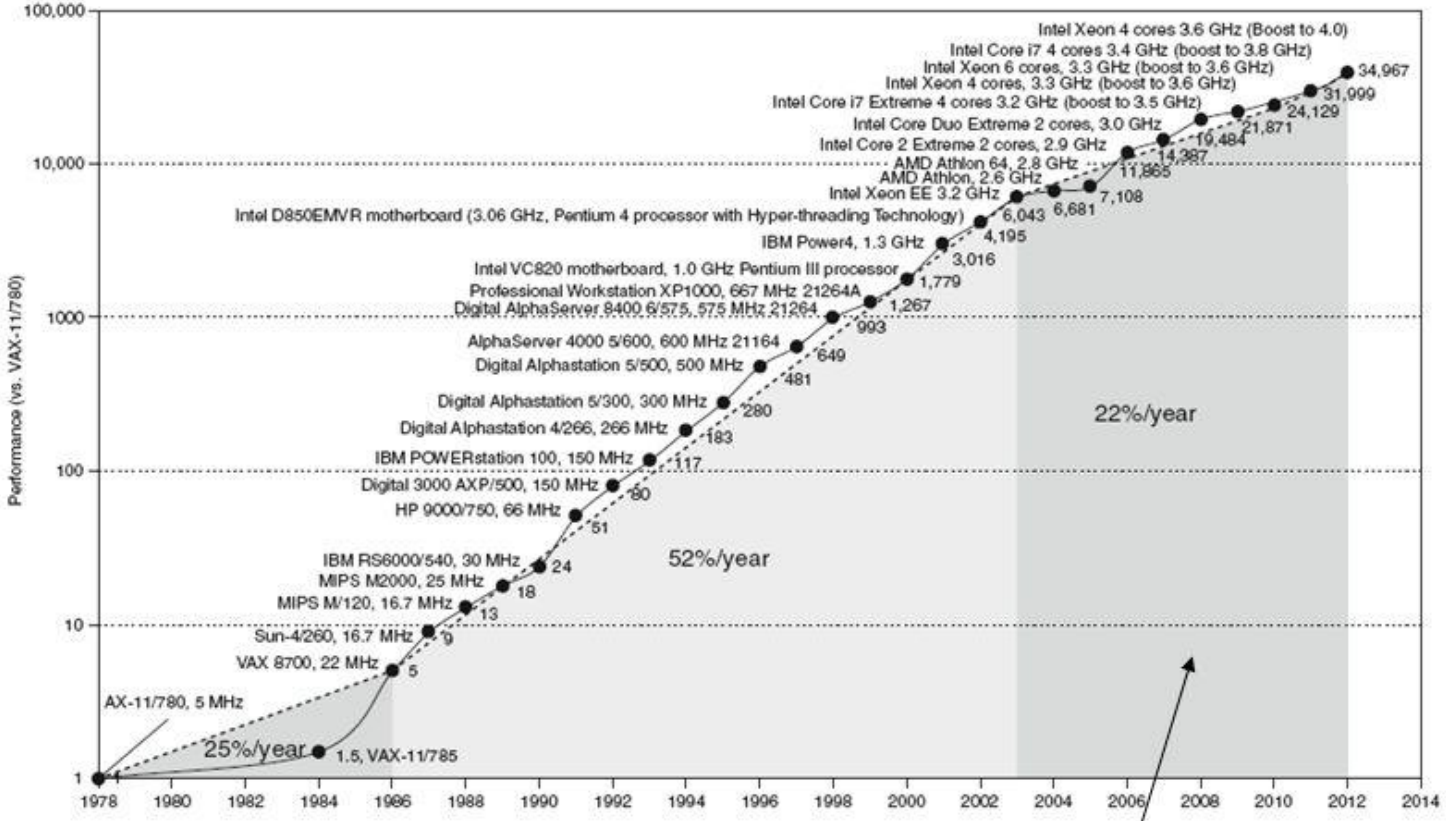
- Sequence 1: IC (number of instructions) = 5
  - CPU Clock Cycles = $2 \times 1 + 1 \times 2 + 2 \times 3 = 10$
  - Avg. CPI = $10/5 = 2.0$
- Sequence 2: IC = 6
  - CPU Clock Cycles = $4 \times 1 + 1 \times 2 + 1 \times 3 = 9$
  - Avg. CPI = $9/6 = 1.5$

Performance Summary

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

- **Performance depends on**
  - Algorithm: affects Number of Instructions, possibly CPI
  - Programming language: affects Number of Instructions, CPI
  - Compiler: affects Number of Instructions, CPI
  - Instruction set architecture: affects Number of Instructions, CPI, T_c
- **Factors are interdependent**
  - RISC: increases instructions/program, but decreases CPI& clock cycle time because the instructions are simple
  - CISC: decreases instructions/program, but increases CPI & clock cycle time because many instructions are more complex

Uniprocessor Performance



Constrained by power, instruction-level parallelism, memory latency

§1.8 The Sea Change: The Switch to Multiprocessors

Multiprocessors

- Multicore microprocessors
  - More than one processor per chip
- Requires explicitly parallel programming
  - Compare with instruction level parallelism
    - Hardware executes multiple instructions at once
    - Hidden from the programmer
  - Hard to do
    - Programming for performance
    - Load balancing
    - Optimizing communication and synchronization

Benchmarks

- A set of programs used to measure performance.
- Performance best determined by running a real application
  - Use programs typical of expected workload
  - Or, typical of expected class of applications e.g., compilers/editors, scientific applications, graphics, etc.

Small benchmarks

- Easier for architects and designers
- easy to standardize than large programs
- can be abused

Benchmarks

- **SPEC CPU Benchmark**
  - ❑Programs used to measure performance
  - ❑Supposedly typical of actual workload
- **Standard Performance Evaluation Corp (SPEC)**
  - ❑Develops benchmarks for CPU, I/O, Web, …
- **SPEC CPU2006** (https://www.spec.org/cpu2006/)
  - ✓Elapsed time to execute a selection of programs
    - ❑Negligible I/O, so focuses on CPU performance
  - ✓Normalize relative to reference machine
  - ✓Summarize as geometric mean of performance ratios
    - ❑CINT2006 (integer) and CFP2006 (floating-point)
- **SPEC CPU 2017** (https://www.spec.org/cpu2017/)

Amdahl's Law

- Performance improvement from speeding up a part of a computer system is limited by the proportion of time the enhancement is used.
- Speed-up = Execution time for entire task without enhancement/Execution time for entire task using enhancement
- Speed-up = Performance for entire task using enhancement/performance for entire task without enhancement
- Execution Time After Improvement = Execution Time Unaffected +(Execution Time Affected / Amount of Improvement)
- **Example:** Suppose a program runs in 100 seconds on a machine, with multiply responsible for 80 seconds of this time. How much do we have to improve the speed of multiplication if we want the program to run four times faster?
- **Solution:** 25 sec = 20 sec + 80 sec/speed

Pitfall: Amdahl's Law

- Improving an aspect of a computer and expecting a proportional improvement in overall performance

$$T_{improved} = \frac{T_{affected}}{\text{improvement factor}} + T_{unaffected}$$

- **Example:** multiply accounts for 80s/100s
  - How much improvement in multiply performance to get 5× overall?

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

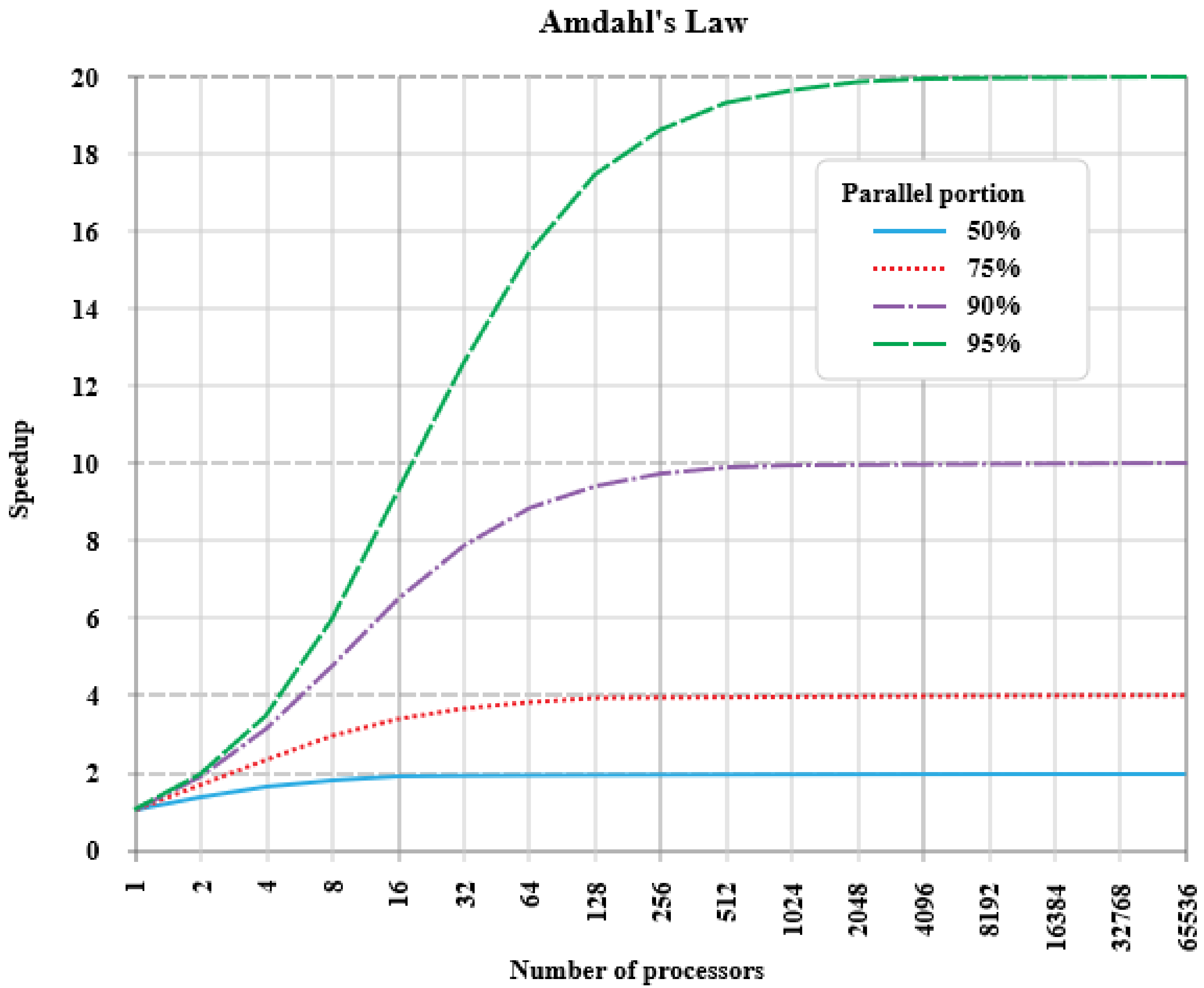
  - Can't be done!

- Corollary: make the common case fast

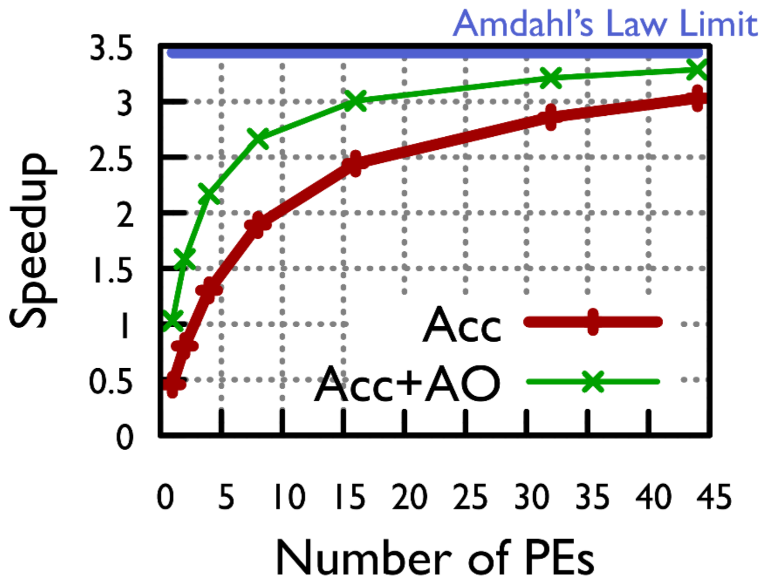
Amdahl's Law

- Overall Speedup $= \dfrac{1}{(1-F)+\dfrac{F}{S}}$
  - **F =** The fraction enhanced
  - **S =** The speedup of the enhanced fraction
- **Example:**
  - Overall speedup if we make 90% of a program run 10 times faster
    - F = 0.9 , S = 10
    - Overall Speedup $= \dfrac{1}{(1-0.9)+\dfrac{0.9}{10}} = 5.26$
  - Overall speedup if we make 80% of a program run 20% faster.
    - F = 0.8 , S = 1.2
    - Overall Speedup $= \dfrac{1}{(1-0.8)+\dfrac{0.8}{1.2}} = 1.153$

Amdahl’s Law



Amdahl's Law
Parallel portion
50%
75%
90%
95%
Speedup
0
2
4
6
8
10
12
14
16
18
20
1
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65536
Number of processors



Amdahl’s Law Limit
Speedup
0
0.5
1
1.5
2
2.5
3
3.5
Acc
Acc+AO
0
5
10
15
20
25
30
35
40
45
Number of PEs

Pitfall: MIPS as a Performance Metric

- MIPS : Millions of instructions per second
- MIPS = Instruction count/(execution time x 10^6) = Clock rate/(CPI x 10^6) → CPI = Instruction count x CPI/Clock rate
- Instruction set-dependent
- Complier technology-dependent
- Program-dependent
- Intuitive: the higher is the better

Pitfall: MFLOPS as a Performance Metric

- MFLOPS: Millions of floating point operations per second
- MFLOPS: Floating point operations /(execution time x 10^6)
- Different machines implement different FP operations
- Different FP operations take different amounts of time
- Only measures FP code

Concluding Remarks

- **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