컴퓨터 공학 개론
Lecture 3

2017

김태성

CPI in More Detail

- **If different instruction classes take different numbers of cycles**

$$\text{Clock Cycles} = \sum_{i=1}^{n} (\text{CPI}_i \times \text{Instruction Count}_i)$$

  - Weighted average CPI

Relative frequency

$$\text{CPI} = \frac{\text{Clock Cycles}}{\text{Instruction Count}} = \sum_{i=1}^{n} \left(\text{CPI}_i \times \frac{\text{Instruction Count}_i}{\text{Instruction Count}} \right)$$

  - **Example**

Instr. Class	Freq	CPI_i		% Time
A	50%	1	.5	33%
B	20%	2	.4	27%
C	10%	2	.2	13%
D	20%	2	.4	27%

CPI = 1.5

CPI Example

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

Instruction Class	A	B	C
CPI for class	1	2	3

Code sequence	Instruction Counts (IC)		
	A	B	C
1	2	1	2
2	4	1	1

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

Performance Summary

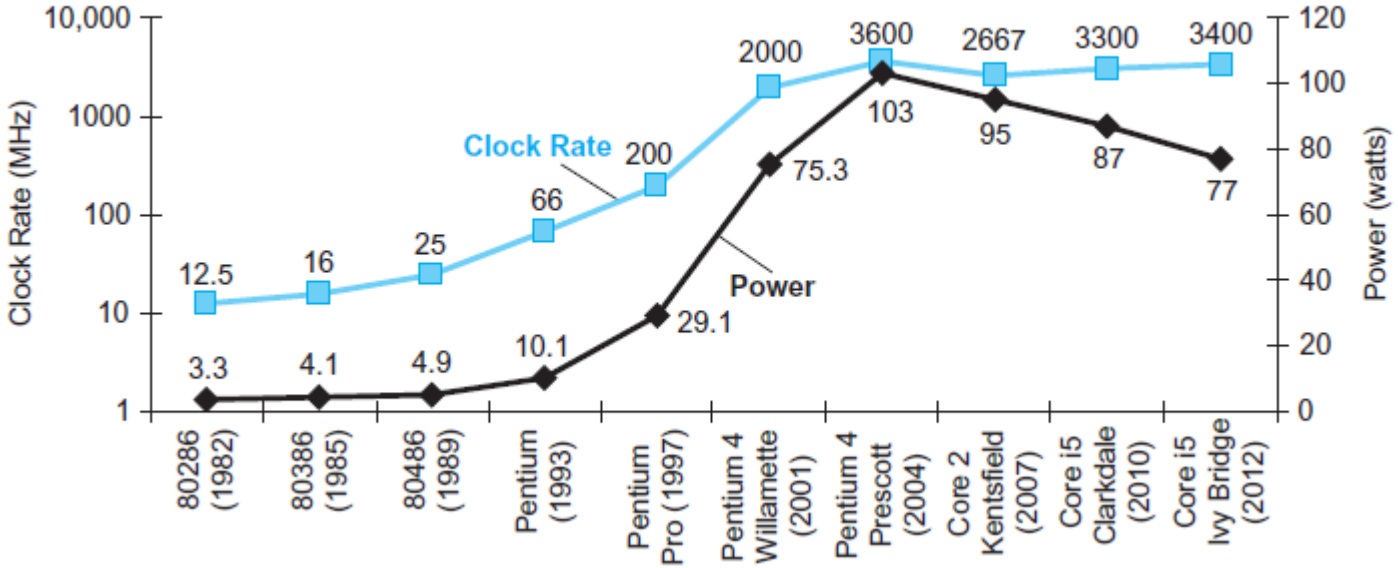
The BIG Picture

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

Instruction Counts(IC) CPI Clock Period(Tc)

- **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 Trends



° **In CMOS IC technology**

$$\text{Power} = \text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency}$$

×30

5V → 1V

×300

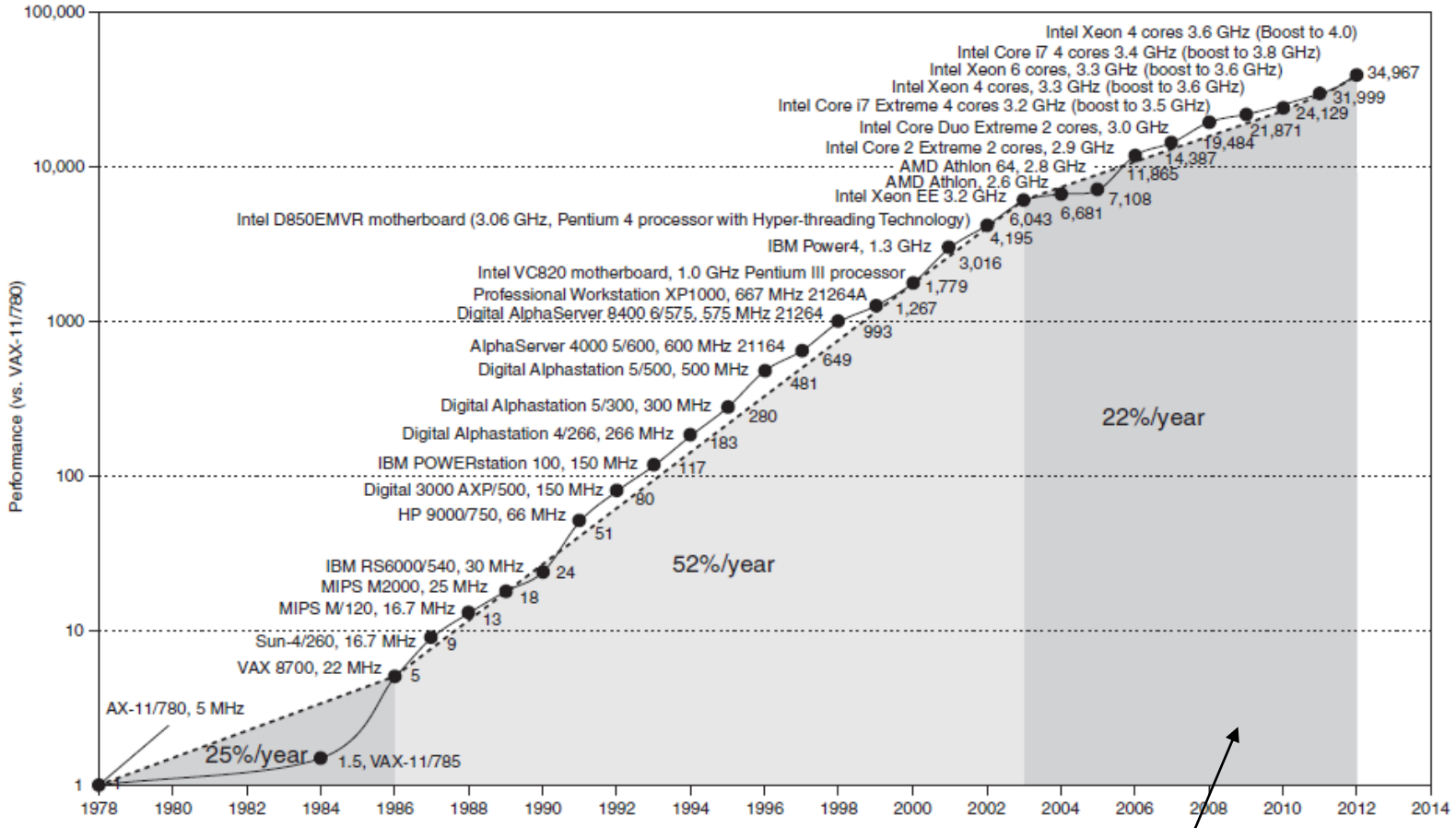
Reducing Power

- **Suppose a new CPU has**
  - **85% of capacitive load of old CPU**
  - **15% voltage and 15% frequency reduction**

$$\frac{P_{new}}{P_{old}} = \frac{C_{old} \times 0.85 \times (V_{old} \times 0.85)^2 \times F_{old} \times 0.85}{C_{old} \times V_{old}^{\ 2} \times F_{old}} = 0.85^4 = 0.52$$

- The power wall
  - We can't reduce voltage further
- How else can we improve performance?

Uniprocessor Performance



Constrained by power, memory latency, etc.

Multiprocessors

- Multicore microprocessors
  - More than one processor per chip
  - Increase on throughput rather than on response time

Product	AMD Opteron X4 (Barcelona)	Intel Nehalem	IBM Power 6	Sun Ultra SPARC T2 (Niagara 2)
Cores per chip	4	4	2	8
Clock rate	2.5 GHz	~2.5 GHz?	4.7 GHz	1.4 GHz
Microprocessor power	120 W	~100 W ?	~100 W ?	94 W

- Requires explicitly parallel programming
  - Hardware executes multiple instructions at once
  - Hidden from the programmer
- Hard to do
  - Programming for performance
  - Load balancing
  - Optimizing communication and synchronization

‘SPEC’ CPU Benchmark

- **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**
  - **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**

$$\sqrt[n]{\prod_{i=1}^{n} \text{Execution time ratio}_i}$$

SPECINT2006* for AMD X4 (Barcelona)

Name	Description	IC×10^9	CPI	Tc (ns)	Exec time	Ref time	SPECratio
perl	Interpreted string processing	2,118	0.75	0.40	637	9,777	15.3
bzip2	Block-sorting compression	2,389	0.85	0.40	817	9,650	11.8
gcc	GNU C Compiler	1,050	1.72	0.40	724	8,050	11.1
mcf	Combinatorial optimization	336	10.00	0.40	1,345	9,120	6.8
go	Go game (AI)	1,658	1.09	0.40	721	10,490	14.6
hmmer	Search gene sequence	2,783	0.80	0.40	890	9,330	10.5
sjeng	Chess game (AI)	2,176	0.96	0.40	837	12,100	14.5
libquantum	Quantum computer simulation	1,623	1.61	0.40	1,047	20,720	19.8
h264avc	Video compression	3,102	0.80	0.40	993	22,130	22.3
omnetpp	Discrete event simulation	587	2.94	0.40	690	6,250	9.1
astar	Games/path finding	1,082	1.79	0.40	773	7,020	9.1
xalancbmk	XML parsing	1,058	2.70	0.40	1,143	6,900	6.0
Geometric mean							**11.7**

SPECINT2006: for integer operations

SPEC Power Benchmark

- **Power consumption of server at different workload levels**
  - **Performance: ssj_ops/sec**
  - **Power: Watts (Joules/sec)**

$$\text{Overall ssj\_ops per Watt} = \left(\sum_{i=0}^{10} \text{ssj\_ops}_i\right) \Big/ \left(\sum_{i=0}^{10} \text{power}_i\right)$$

*ssj: server side Java

SPECpower_ssj2008 for X4

Target Load %	Performance (ssj_ops/sec)	Average Power (Watts)
100%	231,867	295
90%	211,282	286
80%	185,803	275
70%	163,427	265
60%	140,160	256
50%	118,324	246
40%	920,35	233
30%	70,500	222
20%	47,126	206
10%	23,066	180
0%	0	141
Overall sum	1,283,590	2,605
∑ssj_ops/ ∑power		493

Pitfall

- **Improving an aspect of a computer and expecting a proportional improvement in overall performance**
- **Amdahl's Law**
  - **The performance enhancement possible with a given improvement is limited by the amount that the improved feature is used.**

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

- Example: multiply accounts for 80s out of 100s
  - How much improvement will be made if multiply performance gets 5×?

$$T_{improved} = \frac{80}{5} + 20 = 36 \Rightarrow \frac{100}{36} = 2.78$$

- **Lesson: make the common case fast!**

Fallacy: Low Power at Idle

- Look back at X4 power benchmark
  - At 100% load: 295W
  - At 50% load: 246W (83%)
  - At 10% load: 180W (61%)
- Google data center
  - Mostly operates at 10% – 50% load
  - At 100% load less than 1% of the time
- Consider designing processors to make power proportional to load

Pitfall: MIPS as a Performance Metric

- **MIPS: Millions of Instructions Per Second**
  - **Doesn't account for**
    - **Differences in ISAs between computers**
    - **Differences in complexity between instructions**

$$\text{MIPS} = \frac{\text{Instruction count}}{\text{Execution time} \times 10^6}$$

$$= \frac{\text{Instruction count}}{\frac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}} \times 10^6} = \frac{\text{Clock rate}}{\text{CPI} \times 10^6}$$

- CPI varies between programs on a given CPU

Concluding Remarks

- Cost/performance is improving
  - Due to underlying technology development
- Hierarchical layers of abstraction
  - High-level language
  - Assembly language
  - Machine language (hardware representation)
- Execution time: the best performance measure
- Power is a limiting factor
  - Use parallelism to improve performance