CS202 : COMPUTER ORGANIZATION

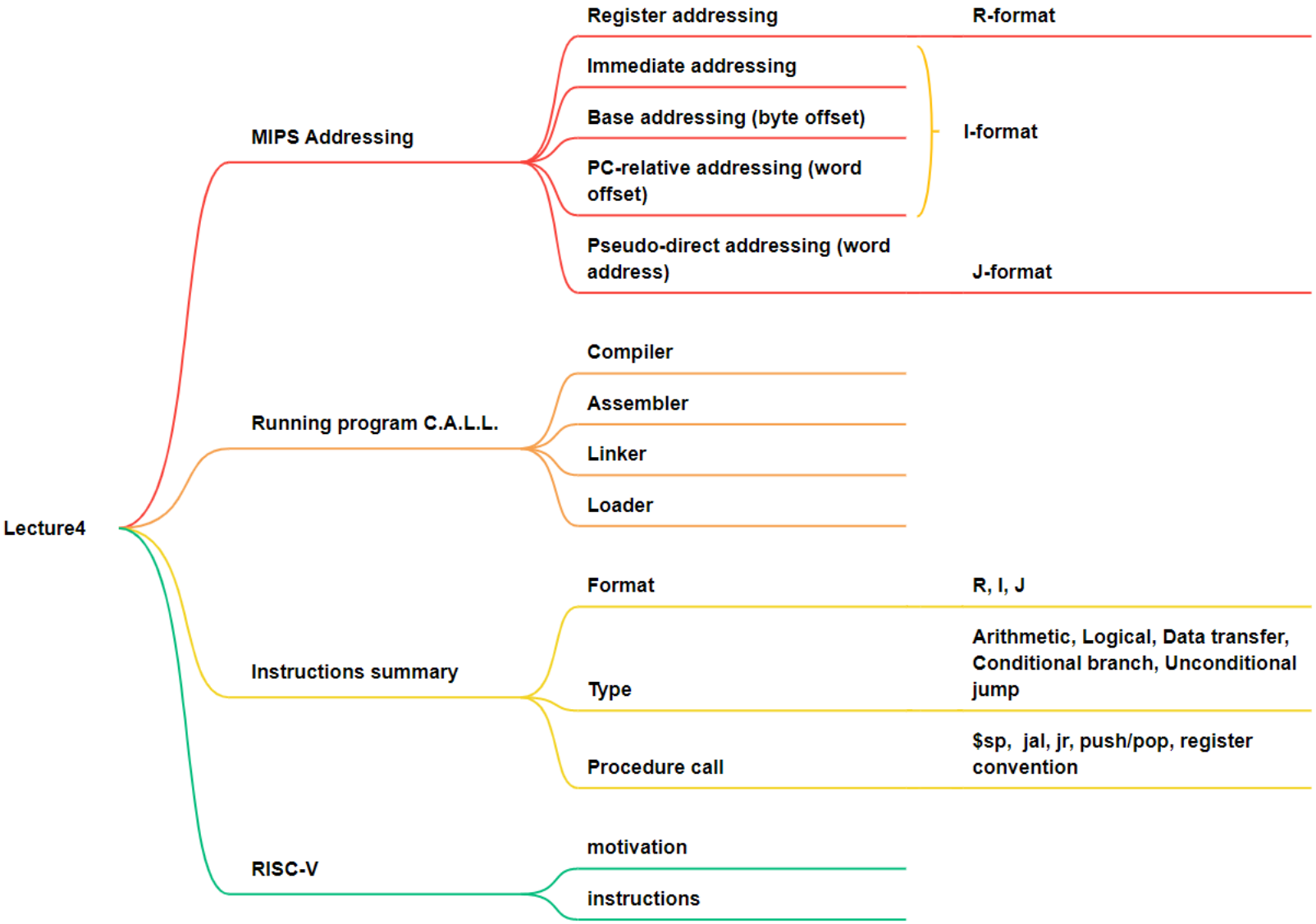
Lecture 5
Performance

2023 Spring

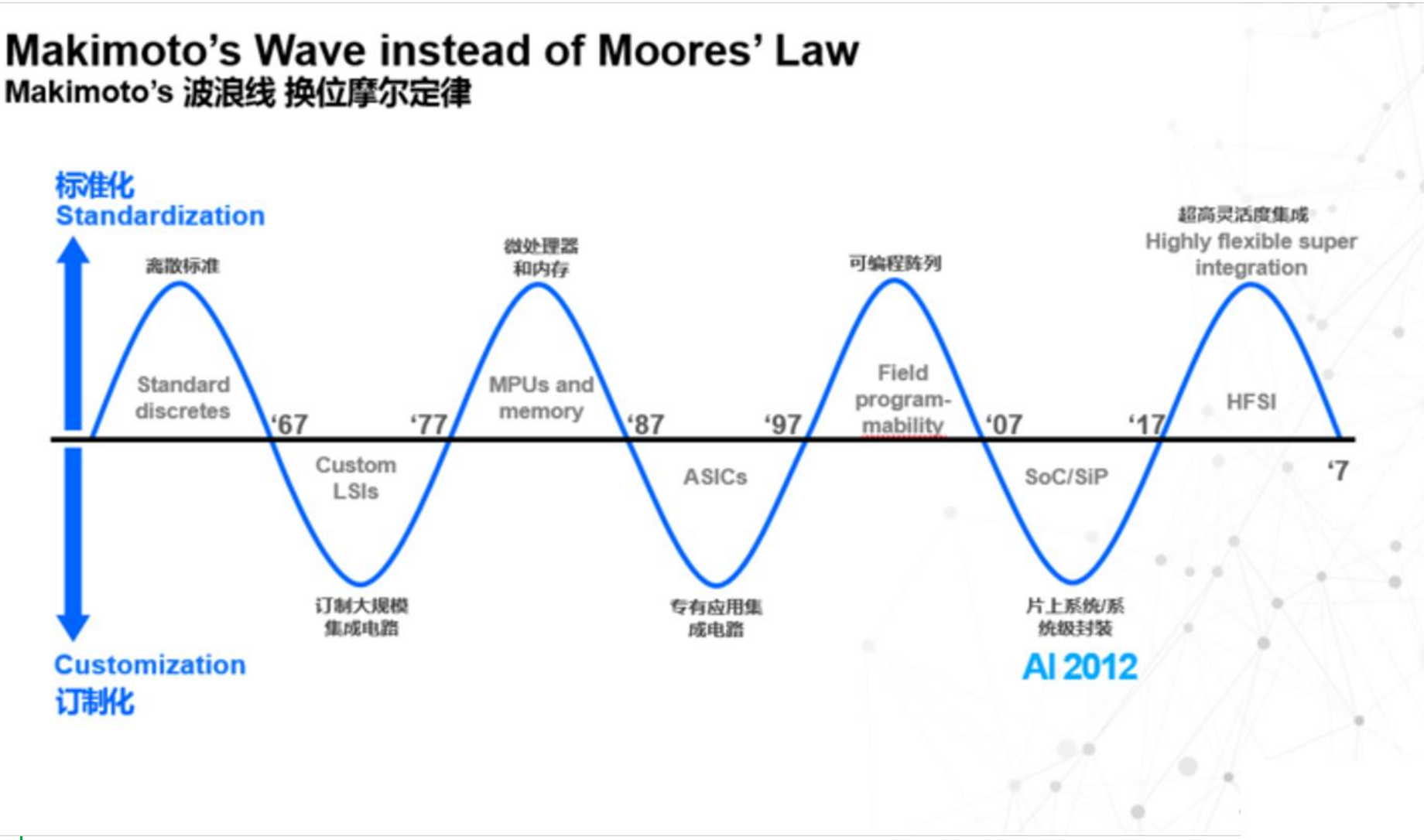
Today's Agenda

- Recap
- Context
  - Performance
    - Definition
    - CPU performance
  - Power wall
  - Measuring and evaluating performance
- Reading: Textbook, Sections 1.6 - 1.12

Recap

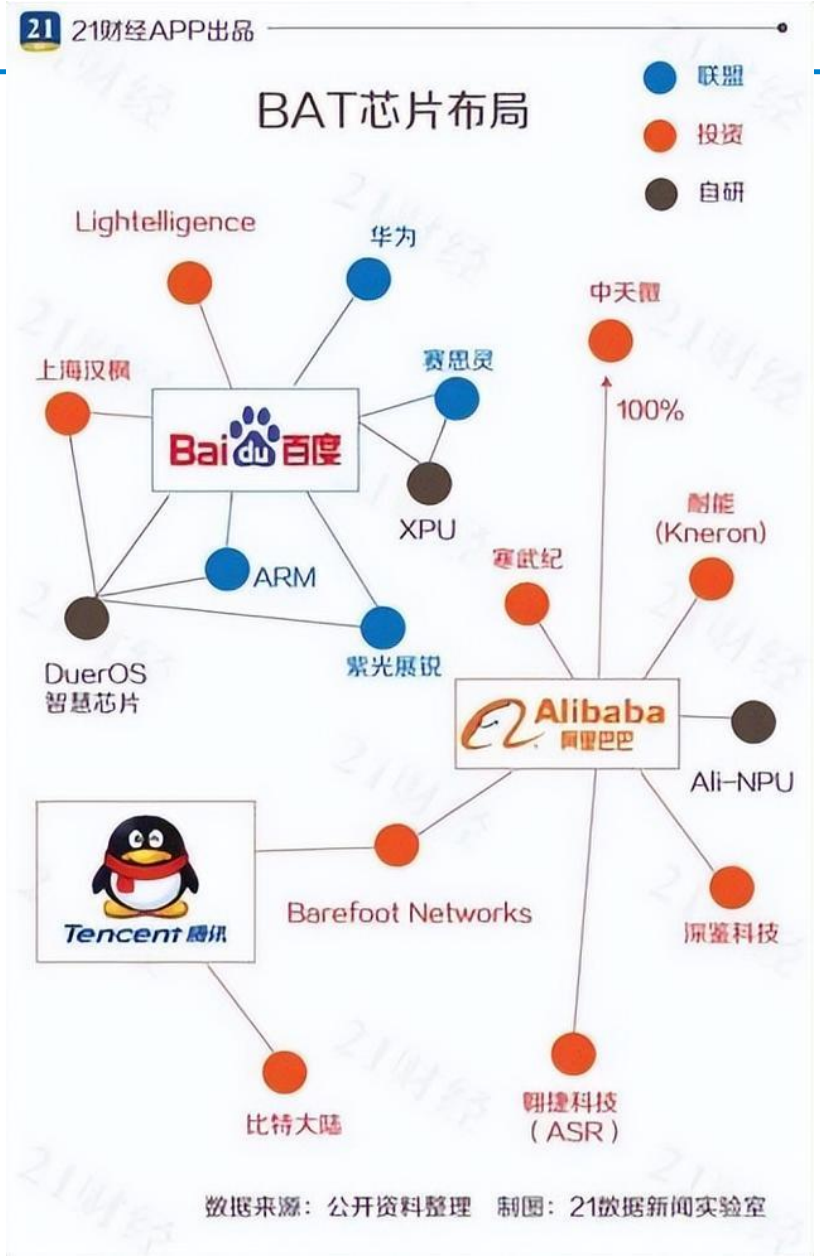

Lecture4
MIPS Addressing
Register addressing
R-format
Immediate addressing
Base addressing (byte offset)
PC-relative addressing (word offset)
I-format
Pseudo-direct addressing (word address)
J-format
Running program C.A.L.L.
Compiler
Assembler
Linker
Loader
Instructions summary
Format
R, I, J
Type
Arithmetic, Logical, Data transfer, Conditional branch, Unconditional jump
Procedure call
$sp, jal, jr, push/pop, register convention
RISC-V
motivation
instructions

Makimoto’s Wave



RISC-V motivations

- This decade: HFSI
  - AI acceleration
  - IoT application specified
- AI inferencing, edge computing
  - GPU (general purpose)
  - Google TPU (application specified)
- RISC-V
  - free → instruction extension → application specified → contributions in ISA/tools/software → ecosystem



Alibaba XuanTie C906

- MLPerf Tiny Benchmark winner (April 2022)
- XuanTie C906 -- based on RISC-V

https://mlcommons.org/

Menu

April 06, 2022 - Inference: Tiny

v0.7 Results

Other Rounds

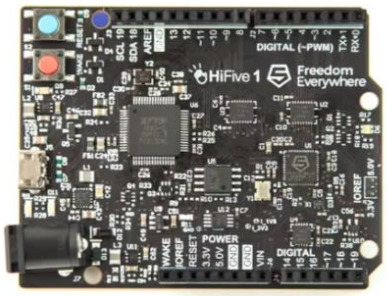
Inference Tiny v0.7

Closed | Open

									Results				
									Task	Visual Wake Words		Image Classific	
									Data	Visual Wake Words Dataset		CIFAR-10	
ID	Submitter	Board	SoC	Processor(s)	Accelerator(s)	Software	Model Used	Accuracy	Units	Latency in ms	Energy in uJ	Latency in ms	En uJ
CATEGORY: Available													
0.7-2015	Alibaba	Nezha	D1	XuanTie C906 RISC-V x 1		Sinian/CSI-NN2	mobilenet_sinian	80.3%		4.702			
0.7-2016	Alibaba	Nezha	D1	XuanTie C906 RISC-V x 1		Sinian/CSI-NN2	resnet_sinian	86.0%				3.122	
0.7-2017	Alibaba	Nezha	D1	XuanTie C906 RISC-V x 1		Sinian/CSI-NN2	dscnn_sinian	90.70%					
0.7-2018	Alibaba	Nezha	D1	XuanTie C906 RISC-V x 1		Sinian/CSI-NN2	autoencoder_sinian	0.88 AUC					
0.7-2019	hls4ml-FINN	Pynq-Z2	Zynq-7020	Dual Core ARM, Cortex-A9 MPCore	Zynq XC7Z020	hls4ml	Conv2D	83.5%				27.3	
0.7-2020	hls4ml-FINN	Pynq-Z2	Zynq-7020	Dual Core ARM, Cortex-A9 MPCore	Zynq XC7Z020	hls4ml	MLP	0.83 AUC					
0.7-2021	hls4ml-FINN	Pynq-Z2	Zynq-7020	Dual Core ARM, Cortex-A9 MPCore	Zynq XC7Z020	FINN	Conv2D	84.5%				1.5	
0.7-2022	hls4ml-FINN	Pynq-Z2	Zynq-7020	Dual Core ARM, Cortex-A9 MPCore	Zynq XC7Z020	FINN	MLP	82.5%					
0.7-2023	hls4ml-FINN	Arty A7-100T	Artix-100T	MicroBlaze (custom)	Artix-7 C7A100T	hls4ml	Conv2D	83.5%				33.1	
0.7-2024	hls4ml-FINN	Arty A7-100T	Artix-100T	MicroBlaze (custom)	Artix-7 C7A100T	hls4ml	MLP	0.83 AUC					

RISC-V in AI and IoT

- Alibaba Xuantie 910: designed for AI and machine learning applications.
- Greenwaves Technologies GAP9: designed for edge computing applications, ultra low power
- HiFive1: Arduino-compatible microcontroller, ideal for IoT applications.



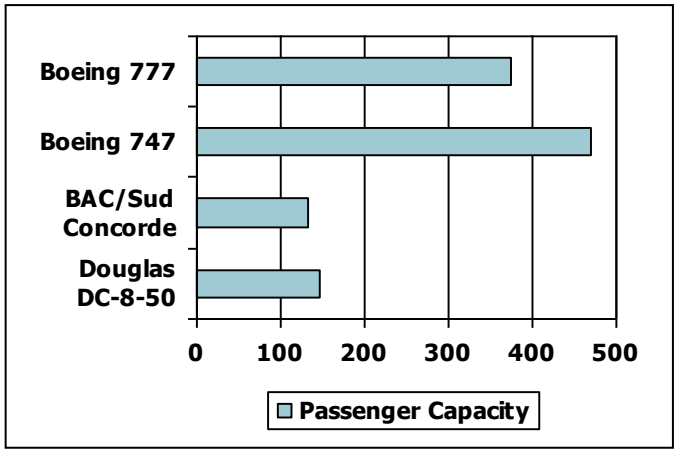
Chapter 1 cont. Performance

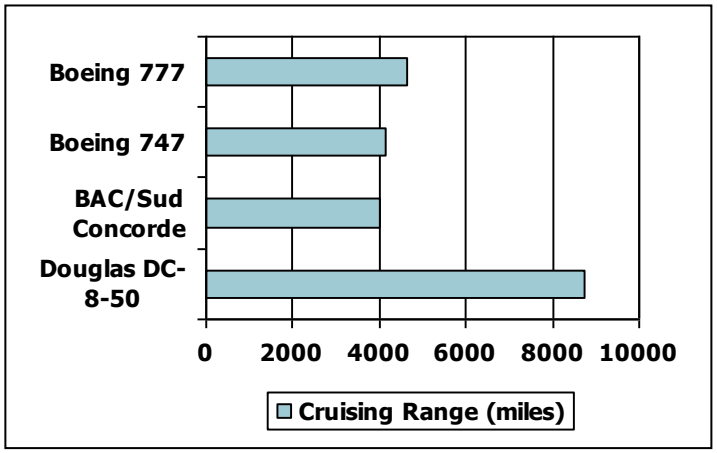
Computer Performance

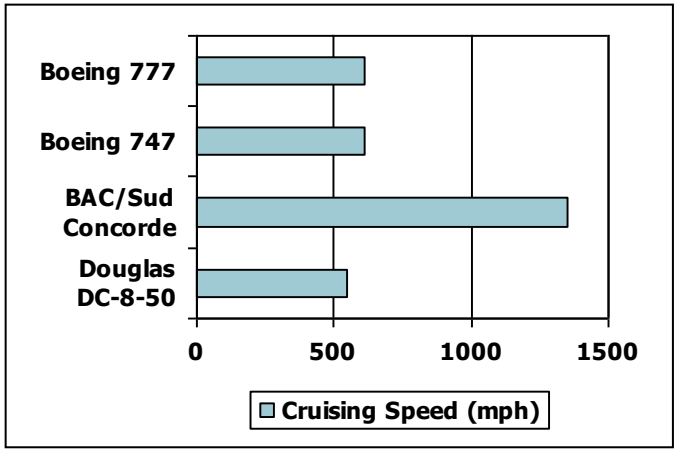
- Computer Performance
- Program Execution Time
- Power Trends
- Amdahl's Law
- Instruction Sets
- Basic Arithmetic Instructions

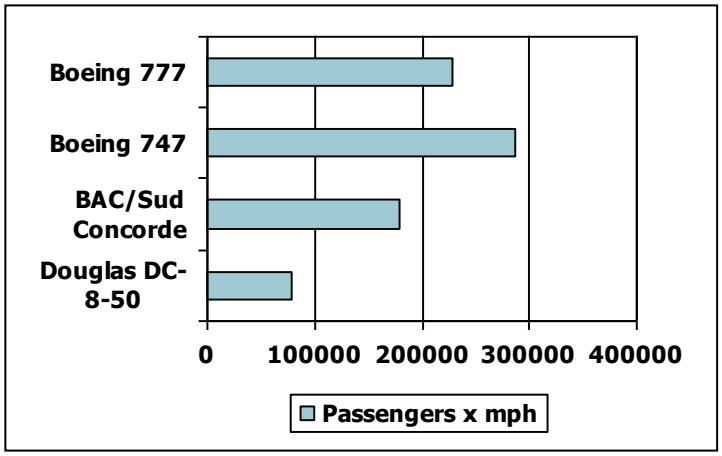
Defining Performance

- Which airplane has the best performance?
  - To evaluate the performance, we must define the metric first!









Computer Performance

- Performance of a computer is based on the following criteria:
- Response time
  - Elapsed time between the start and the end of one task
  - i.e. How long it takes to do a task
- Throughput
  - Total number of tasks finished in a given interval of time.
  - e.g., tasks/transactions/… per hour
- How are response time and throughput affected by
  - Replacing the processor with a faster version?
  - Adding more processors?
- We'll focus on response time for now…

Relative Performance

- Defining performance:

$$\text{Performance} = \frac{1}{\text{Execution time}}$$

- What does “X is n times faster than Y” mean?

$$\text{Performance ratio (n)} = \frac{\text{Performance}_x}{\text{Performance}_y} = \frac{\text{Execution time}_y}{\text{Execution time}_x}$$

- Example:
  - 10s on y, 15s on x for the same program running

$$\frac{\text{Execution time}_y}{\text{Execution time}_x} = \frac{15\text{s}}{10\text{s}} = 1.5$$

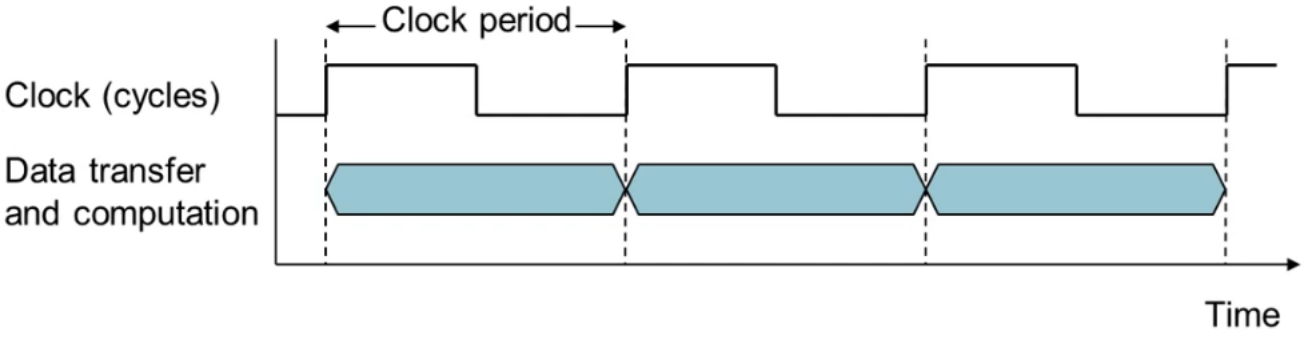
  - So x is 1.5 times *faster* than y, or y is 1.5 times *slower* than x
- Increase performance = decrease execution time
  - “**improve**” performance/execution time

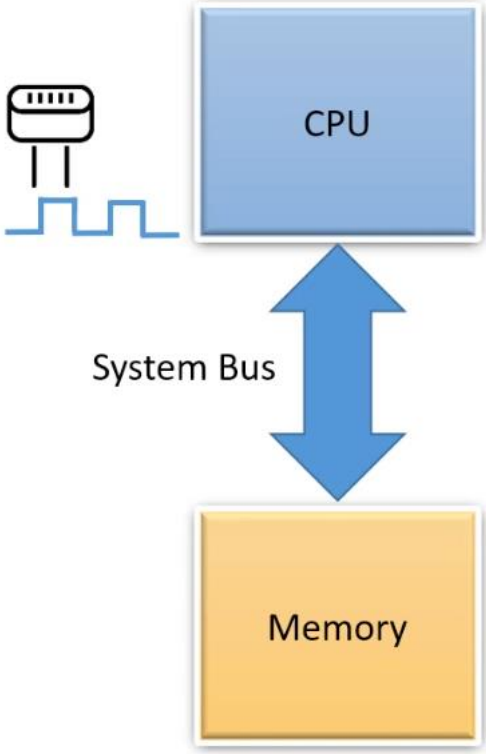
Measuring Execution Time

- Elapsed time
  - Total response time, including all aspects
    - Processing, I/O, OS overhead, idle time
  - Determines system performance
- CPU time
  - Time spent processing a specific task
    - Minus I/O time, other jobs' shares
    - Includes user CPU time and system CPU time
  - Different programs are affected differently by CPU and system performance
    - Running on servers-I/O performance-hardware and software
    - Total elapsed time is of interest
    - Define performance metric and then proceed

CPU Clocking

- Operation of digital hardware governed by a constant-rate clock
- Notions: Clock cycle, Clock period, Clock rate/frequency





- Clock period (Clock cycle time): duration of a clock cycle
  - e.g., 250ps = 250×10^{-12}s = 0.25×10^{-9}s
- Clock frequency (rate): cycles per second
  - e.g., 4.0GHz = 4000MHz = 4.0×10^{9}Hz
- Clock cycle time = 1/Clock rate

CPU Time

$$\text{CPU Time} = \text{No. of Clock Cycles} \times \text{Clock Period}$$

$$= \frac{\text{No. of Clock Cycles}}{\text{Clock Rate}}$$

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

CPU Time Example

- Computer A: 2GHz clock, 10s CPU time
- Designing Computer B
  - Aim for 6s CPU time with 1.2 × No. of clock cycles
- How fast must Computer B clock be?

$$\text{CPU Time} = \frac{\text{Clock Cycles}}{\text{Clock Rate}}$$

$$\text{Clock Rate}_B = \frac{\text{Clock Cycles}_B}{\text{CPU Time}_B} = \frac{1.2 \times \text{Clock Cycles}_A}{6\text{s}}$$

$$\begin{aligned}\text{Clock Cycles}_A &= \text{CPU Time}_A \times \text{Clock Rate}_A \\ &= 10\text{s} \times 2\text{GHz} = 20 \times 10^9\end{aligned}$$

$$\text{Clock Rate}_B = \frac{1.2 \times 20 \times 10^9}{6\text{s}} = \frac{4 \times 10^9 \text{cycles}}{\text{s}} = 4\text{GHz}$$

Instruction Count and CPI

$$\text{Clock Cycles} = \text{Instruction Count} \times \text{Cycles per Instruction(CPI)}$$

$$\text{CPU Time} = \text{Instruction Count} \times \text{CPI} \times \text{Clock Period}$$

$$= \frac{\text{Instruction Count} \times \text{CPI}}{\text{Clock Rate}}$$

- CPU executes instructions sequentially
  - Instruction fetch, decode, execution, memory access, write back
- Instruction Count for a program
  - Determined by program, ISA, and compiler
- Clock cycles per instruction (CPI)
  - Average cycles per instruction: **No. of cycles/Instruction Count**
  - Determined by CPU hardware
  - If different instructions have different CPI
    - Average CPI gets affected by instruction mix

CPI Example

- Computer A: Cycle Time = 250ps, CPI = 2.0
- Computer B: Cycle Time = 500ps, CPI = 1.2
- Same ISA
- Which is faster, and by how much?

$$\frac{\text{Performance}_x}{\text{Performance}_y} = \frac{\text{Execution time}_y}{\text{Execution time}_x} = n$$

$$\text{CPU Time} = \text{Instructions} \times \text{CPI} \times \text{Clock Period} = \frac{\text{Instructions} \times \text{CPI}}{\text{Clock Rate}}$$

$$\text{CPU Time}_A = \text{Instruction Count} \times \text{CPI}_A \times \text{Clock Period}_A = I \times 2.0 \times 250\text{ps} = I \times 500\text{ps}$$

$$\text{CPU Time}_B = \text{Instruction Count} \times \text{CPI}_B \times \text{Clock Period}_B = I \times 1.2 \times 500\text{ps} = I \times 600\text{ps}$$

$$\frac{\text{CPU performance}_A}{\text{CPU performance}_B} = \frac{\text{Execution time}_B}{\text{Execution time}_A} = \frac{I \times 600\text{ps}}{I \times 500\text{ps}} = 1.2$$

- So, computer A is 1.2 times as fast as computer B

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

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

Relative frequency

Compiler design - Code Segments

- Alternative compiled code sequences using instructions in classes A, B, C. Instruction counts? CPI? Which is faster?

Class	A	B	C	Instruction counts	Clock cycles	CPI
CPI for class	1	2	3	/	/	/
IC in sequence 1	2	1	2	**5**	10	2
IC in sequence 2	4	1	1	6	**9**	1.5

- Sequence 1: IC = 5
- Clock Cycles
  = 2x1+1x2+2x3
  =10
- Avg.CPI = 10/5 = 2.0

- Sequence 2: IC = 6
- Clock Cycles
  = 4x1+1x2+1x3
  =9
- Avg.CPI = 9/6 = 1.5

- Application: compiler optimization
  - minimize execution time: select sequence 2(EX: GNU gcc -O3)
  - minimize code size: select sequence 1(EX:GNU gcc -Os)

Performance Summary

- **The Classic CPU Performance Equation**

$$\begin{aligned}\text{CPU Time} &= \text{Instruction Count(IC)} \\ &\quad \times \text{Cycles per Instruction(CPI)} \\ &\quad \times \text{Clock Period}(T_c) \\ &= IC \times CPI \times T_c \\ &= IC \times CPI / f \quad (f = 1/T_c)\end{aligned}$$

- Performance depends on

	Instruction Count	CPI	Clock Rate
Algorithm	√	√	
Language	√	√	
Compiler	√	√	
ISA	√	√	√
Organization		√	√
Technology			√

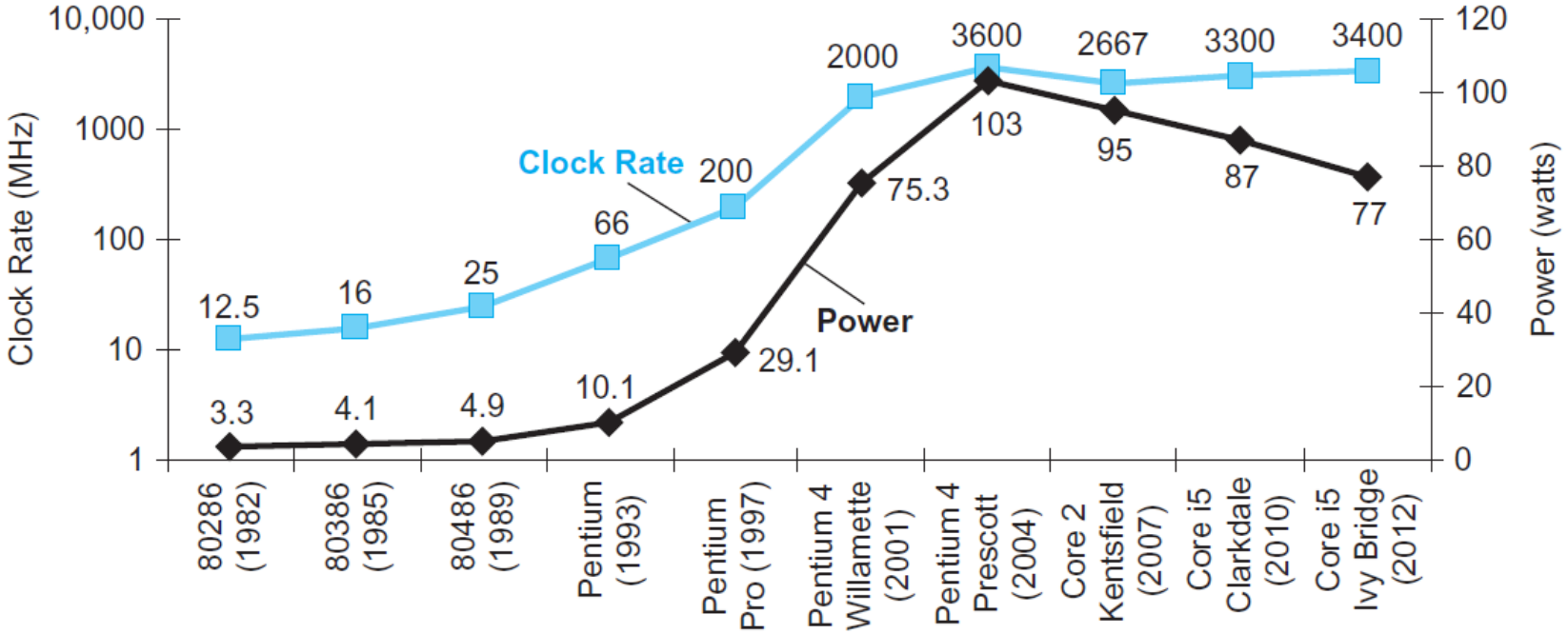
Performance Summary

$$\text{CPU Time} = \text{IC} \times \text{CPI} \times T_c = \frac{\text{IC} \times \text{CPI}}{f}$$

- How are response time and throughput affected by
  - Replacing the processor with a faster version?
  - Adding more processors?
- Usage scenarios:
  - Real-time system: car engine control system, heart pacemakers, video game systems, obstacle avoiding robot
  - Worst-case execution time (WCET): the maximum length of time the task could take to execute on a specific hardware platform
- **Execution time(CPU Time)** is the only **complete** and **reliable** measure of computer performance
  - Using a subset of the performance equation as a performance metric is not reliable, for example MIPS (million instructions per second)!

Power Trends

- One way of enhancing performance is to increase clock rate (implying a reduction in clock cycle time) .
- A consequence of increasing clock rate is an increase in power dissipation
  - Motivation: Reducing CPU Power to increase battery time, reduce cooling cost.



Energy dissipation

- Energy consumption = dynamic energy + static energy
  - Dynamic energy (energy spent when transistors switch from 0→1 1→0) is primary
  - Static energy is the energy cost when no transistor switches
- Energy for 0→1→0:

$$\text{Energy} \propto \text{Capacitive load} \times \text{Voltage}^2$$

- Energy for 0→1 or 1→0 per second (power):

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

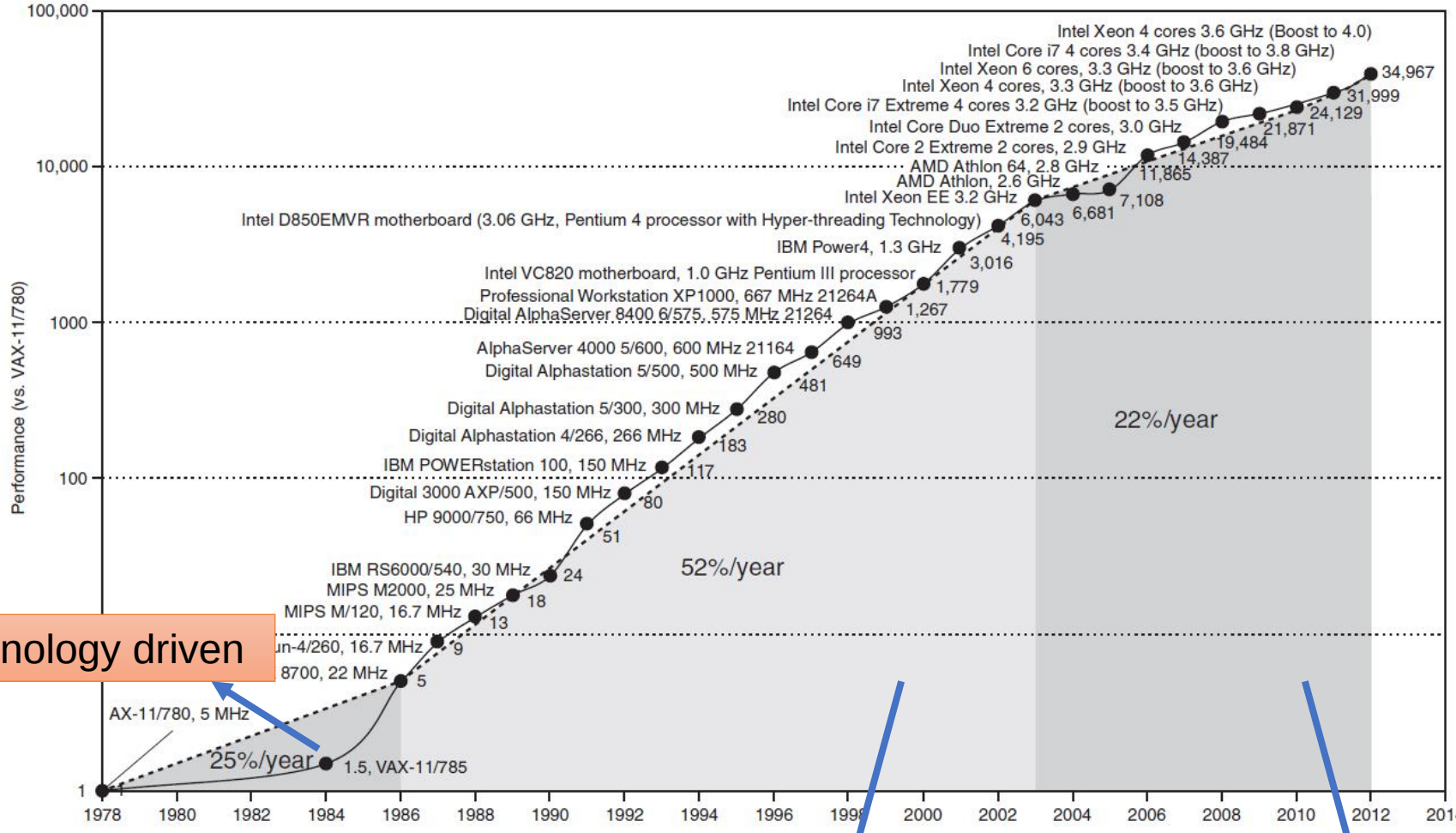
Reducing Power

- 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
  - We can't remove more heat
- How else can we improve performance?

Uniprocessor Performance



Technology driven

Advanced architectural and organizational ideas

Constrained by power, instruction-level parallelism, memory latency

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 (Why?)
  - Programming for performance
  - Load balancing
  - Optimizing communication and synchronization

Amdahl's Law

- Pitfall(陷阱): Improving an aspect of a computer and expecting a proportional improvement in overall performance
- Amdahl's Law:
  - Calculate how much a computation can be sped up by running part of it in parallel

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

  - **make the common case fast**
- Example:
  - Multiply accounts for 80s/100s in a program
  - How much improvement in multiply performance to get 5 X overall?

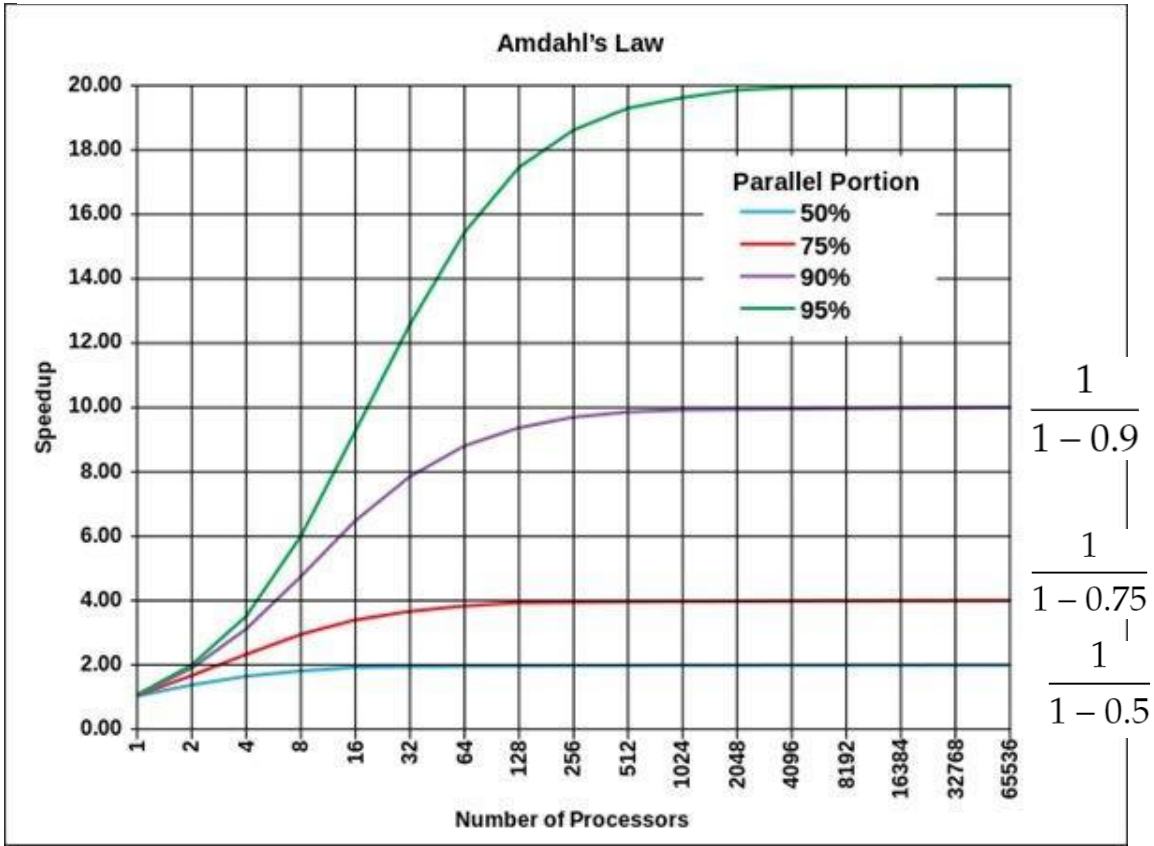
$$\text{Execution time}_{new} = \frac{80s}{n} + (100 - 80\ s) = \frac{80s}{n} + 20s$$

  - Impossible to get 5 times faster overall!

Amdahl's Law

$$\text{Speedup} = \frac{T_1 + T_2}{T'} = \frac{T_1 + T_2}{\frac{T_1}{N} + T_2} = \frac{1}{\frac{1}{N}\left(\frac{T_1}{T_1+T_2}\right) + \left(\frac{T_2}{T_1+T_2}\right)} = \frac{1}{\frac{1}{N}(p) + (1-p)}$$

(p : parallel portion)



$$\frac{1}{1-0.9}$$

$$\frac{1}{1-0.75}$$

$$\frac{1}{1-0.5}$$

Measuring and evaluating performance

- What's wrong with this program as a workload?

```
integer A[][], B[][], C[][];
for (I=0; I<100; I++)
  for (J=0; J<100; J++)
    for (K=0; K<100; K++)
      C[I][J] = C[I][J] + A[I][K]*B[K][J];
```

- What measured? Not measured? What is it good for?
- Ideally run typical programs with typical input before purchase, or before even build machine
  - Called a "workload"; For example:
  - Engineer uses compiler, spreadsheet
  - Author uses word processor, drawing program, compression software

Benchmark

- To compare performance between two computers, we use benchmark
  - Select a set of programs that represent the workload
  - Run these programs on each computer
  - Compare the average execution time (geometric mean)
- Example:
  - Based on the arithmetic and geometric means execution time, which of the two computer is faster?

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

Execution Time	Computer A	Computer B
Program 1	2	10
Program 2	100	105
Program 3	1000	100
Program 4	25	75

$$\frac{\text{Geometric mean}_A}{\text{Geometric mean}_B} = \frac{\sqrt[4]{2 \times 100 \times 1000 \times 25}}{\sqrt[4]{10 \times 105 \times 100 \times 75}} = \sqrt[4]{\frac{2 \times 100 \times 1000 \times 25}{10 \times 105 \times 100 \times 75}} = \sqrt[4]{0.63} < 1$$

  - Computer A is faster

SPEC CPU Benchmark

- Benchmark
  - A set of programs used to assess the performance of a computer
- SPEC: Standard Performance Evaluation Corporation(SPEC CPU2006)
- Benchmarking flow
  - measure execution time
  - normalize to a reference machine
  - summarize as geometric mean of performance ratios
- SPECratio: $$\frac{\text{execution time}_{\text{reference machine}}}{\text{execution time}_{\text{measured machine}}}$$
- Geometric mean:

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

CINT2006 result for Intel Core i7 920

- Integer benchmarks

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

Description	Name	Instruction Count x 10^9	CPI	Clock cycle time (seconds x 10^{-9})	Execution Time (seconds)	Reference Time (seconds)	SPECratio
Interpreted string processing	perl	2252	0.60	0.376	508	9770	19.2
Block-sorting compression	bzip2	2390	0.70	0.376	629	9650	15.4
GNU C compiler	gcc	794	1.20	0.376	358	8050	22.5
Combinatorial optimization	mcf	221	2.66	0.376	221	9120	41.2
Go game (AI)	go	1274	1.10	0.376	527	10490	19.9
Search gene sequence	hmmer	2616	0.60	0.376	590	9330	15.8
Chess game (AI)	sjeng	1948	0.80	0.376	586	12100	20.7
Quantum computer simulation	libquantum	659	0.44	0.376	109	20720	190.0
Video compression	h264avc	3793	0.50	0.376	713	22130	31.0
Discrete event simulation library	omnetpp	367	2.10	0.376	290	6250	21.5
Games/path finding	astar	1250	1.00	0.376	470	7020	14.9
XML parsing	xalancbmk	1045	0.70	0.376	275	6900	25.1
Geometric mean	–	–	–	–	–	–	25.7

SPEC Power Benchmark

- Power consumption of server at different workload levels
  - Performance: ssj_ops (server side Java operations per second)
  - Power: Watts (Joules/sec)

$$\text{overall ssj\_ops per watt} = \left(\sum_{i=0}^{10} \text{ssj}_o\text{ps}_i\right) / \left(\sum_{i=0}^{10} \text{power}_i\right)$$

Target Load %	Performance (ssj_ops)	Average Power (watts)
100%	865,618	258
90%	786,688	242
80%	698,051	224
70%	607,826	204
60%	521,391	185
50%	436,757	170
40%	345,919	157
30%	262,071	146
20%	176,061	135
10%	86,784	121
0%	0	80
Overall Sum	4,787,166	1922
∑ssj_ops / ∑power =		2490

Fallacy: Low Power at Idle

- Fallacy(谬误)
- Look back at i7 power benchmark
  - At 100% load: 258W
  - At 50% load: 170W (66%)
  - At 10% load: 121W (47%)
- Google data center
  - Mostly operates at 10% – 50% load
  - At 100% load less than 1% of time
- Consider designing processors to make power proportional to the load !

Summary

- Knowledge of hardware improves software quality:
  - compilers, OS, threaded programs, memory management
- Important trends:
  - growing density of transistors
  - move to multi-core
  - slowing rate of performance improvement
  - power/thermal constraints
- Reasoning about performance:
  - clock speeds, CPI, benchmark suites, performance equations
- Next:
- assembly instructions