Performance

CSCM601252 - Pengantar Organisasi Komputer

Instructor: Erdefi Rakun dan Tim Dosen POK

Fasilkom UI

Performance and Benchmarking

- Performance Definition
- Factors Affecting Performance
- Measurement Parameters for Performance
- Co-relation Among Performance Parameters
- Benchmarking
- SPEC ’95
- Amdahl’s Law

Details in COD

Note: These slides are taken from Aaron Tan’s slide

Performace Definition (1/5)

- Two perspectives:
  - Purchasing perspective
  - Design perspective
- Performance indices:
  - Which has the best performance?
  - Which has the least cost?
  - Which has best performance/cost?
- Both require:
  - Basis for comparison
  - Metric for evaluation
- Our goal is to understand performance of machine's architectural design.

Performace Definition (2/5)

- Two notions of performance

Air plane	Pa ss enge r capacity	Cr uisi ng ra nge (miles)	Cr uisi ng spe ed (m.p.h.)	Pa ss enge r thr ough put (passen ger s × m.p. h.)
Boeing 777	375	4630	610	228,750
Boeing 747	470	4150	610	286,700
BAC/Sud Co ncorde	132	400 0	1350	178,200
Douglas DC-8-50	146	8720	544	79,424

- Which has higher performance?
  - Time to do ONE task
    - Execution time, response, latency
  - Tasks per day, hour, week, …
    - Throughput, bandwidth.
- Response time and throughput might be in opposition.

Performace Definition (3/5)

- Response time/execution time/latency:
  - Time between start and end of an event
    - How long does it take to execute my job?
    - How long must I wait for the database query?
- Throughput:
  - Total amount of work (or number of jobs) done
    - How many jobs can the machine run at once?
    - What is the average execution rate?

Performace Definition (4/5)

- If we upgrade a machine with a new processor, what do we improve?
  - **Response Time and Throughput**
- If we add a new machine to the lab, what do we improve?
  - **Throughput**

Performace Definition (5/5)

- Performance is in units of things-per-second
  - Bigger is better
- If we are primarily concerned with response time
  - Smaller is better

$$performance_x = \frac{1}{time_x}$$

- "X is *n* times faster than Y" means the speedup *n* is:

$$Speedup = \frac{time_y}{time_x} = \frac{performance_x}{performance_y}$$

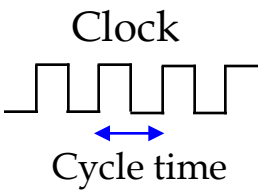
Computing Time (1/3)

- There are different measures of execution time in computer performance.
- Elapsed time
  - Counts everything (including disk and memory accesses, I/O, etc.)
  - Not too good for comparison purposes.
- CPU time
  - Doesn't include I/O or time spent running other programs.
  - Can be broken up into system time and user time.
- Our focus: User CPU time
  - Time spent executing the lines of code in the program

Computing Time (2/3)

- Instead of reporting execution time in seconds, we often use clock cycles (basic time unit in machine).

$$\frac{\text{seconds}}{\text{program}} = \frac{\text{cycles}}{\text{program}} \times \frac{\text{seconds}}{\text{cycle}}$$



- Cycle time (or cycle period or clock period) = time between two consecutive rising edges, measured in seconds.
- Clock rate (or clock frequency) = 1/cycle-time = number of cycles per second (1 Hz = 1 cycle/second).
  - Example: A 200 MHz clock has cycle time of $1/(200 \times 10^6) = 5 \times 10^{-9}$ seconds = 5 nanoseconds.

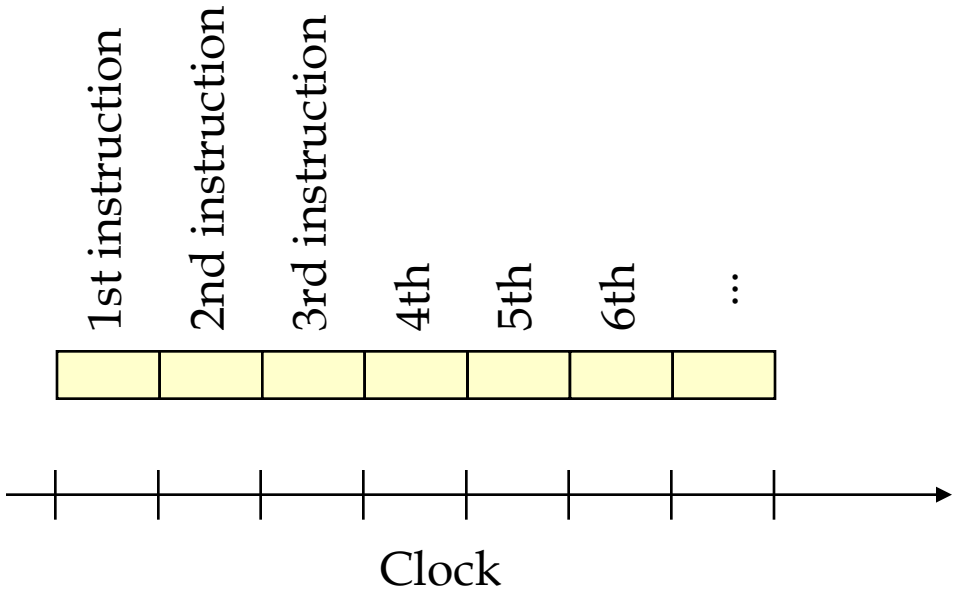
Computing Time (3/3)

$$\frac{\text{seconds}}{\text{program}} = \frac{\text{cycles}}{\text{program}} \times \frac{\text{seconds}}{\text{cycle}}$$

- Therefore, to improve performance (everything else being equal), you can do the following:
  - ↓ Reduce the number of cycles for a program, or
  - ↓ Reduce the clock cycle time, or said in another way,
  - ↑ Increase the clock rate.

Instruction Cycles (1/2)

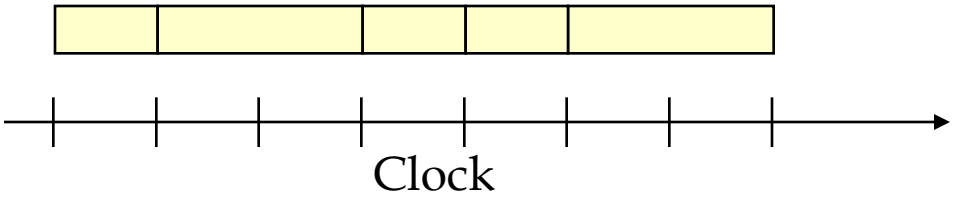
- Can we assume that
  - The number of cycles = number of instructions?
  - The number of cycles is proportional to number of instructions?



- No, the assumptions are incorrect.

Instruction Cycles (2/2)

- Different instructions take different amount of time to finish.



- For example:
  - Multiply instruction may take more cycles than an Add instruction.
  - Floating-point operations take longer than integer operations.
  - Accessing memory takes more time than accessing registers.

Example 1

- Our favorite program runs in 10 seconds on computer A, which has a 400 MHz clock. We are trying to help a computer designer build a new machine B, that will run this program in 6 seconds. The designer can use new (or perhaps more expensive) technology to substantially increase the clock rate, but has informed us that this increase will affect the rest of the CPU design, causing machine B to require 1.2 times as many clock cycles as machine A for the same program. What clock rate should we tell the designer to target at?

- ANSWER:

  Let C be the number of clock cycles required for that program.

  For A: Time = 10 sec. = C × 1/400MHz

  For B: Time = 6 sec. = (1.2 × C) × 1/clock_rateB

  Therefore, clock_rateB = ?

Cycles per Instruction (1/2)

- A given program will require

Some number of instructions (machine instructions)

× **average CPI**

Some number of cycles

× **cycle time**

Some number of seconds

- Recall that different instructions have different number of cycles.

Cycles per Instruction (2/2)

- Average cycle per instruction (CPI)

CPI = (CPU time × Clock rate) / Instruction count

= Clock cycles / Instruction count

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

$$\text{CPI} = \sum_{k=1}^{n} \text{CPI}_k \times F_k \qquad \textbf{where} \qquad F_k = \frac{I_k}{\text{Instruction count}}$$

I_k = instruction frequency

- Invest resources where time is spent!

Example 2

- A compiler designer is deciding between 2 codes for a particular machine. Based on the hardware implementation, there are 3 classes of instructions: Class A, Class B, and Class C, and they require 1, 2, and 3 cycles respectively.
- First code has 5 instructions: 2 of A, 1 of B, and 2 of C.
  Second code has 6 instructions: 4 of A, 1 of B, and 1 of C.
- Which code is faster? By how much?
- What is the (average) CPI for each code?

- ANSWER:

  Let T be the cycle time.
  Time(code1) = (2×1 + 1×2 + 2×3) × T = 10T
  Time(code2) = (4×1 + 1×2 + 1×3) × T = 9T
  Time(code1)/Time(code2) =
  CPI(code1) =
  CPI(code2) =

Example 3

- Suppose we have 2 implementations of the same ISA, and a program is run on these 2 machines.
- Machine A has a clock cycle time of 10 ns and a CPI of 2.0. Machine B has a clock cycle time of 20 ns and a CPI of 1.2.
- Which machine is faster for this program? By how much?

- ANSWER:

  Let N be the number of instructions.

  Machine A: Time = N × 2.0 × 10 ns

  Machine B: Time =

  Performance(A)/Performance(B) = Time(B)/Time(A)

  =

Example 4 (1/4)

- You are given 2 machine designs M1 and M2 for performance benchmarking. Both M1 and M2 have the same ISA, but different hardware implementations and compilers. Assuming that the clock cycle times for M1 and M2 are the same, performance study gives the following measurements for the 2 designs.

Instruction class	For M1		For M2	
	CPI	No. of instructions executed	CPI	No. of instructions executed
A	1	3,000,000,000,000	2	2,700,000,000,000
B	2	2,000,000,000,000	3	1,800,000,000,000
C	3	2,000,000,000,000	3	1,800,000,000,000
D	4	1,000,000,000,000	2	900,000,000,000

Example 4 (2/4)

a) What is the CPI for each machine?

Let Y = 1,000,000,000,000

CPI(M1) = (3Y×1 + 2Y×2 + 2Y×3 + Y×4) / (3Y + 2Y + 2Y + Y)
= 17Y / 8Y = **2.125**

CPI(M2) =
=

b) Which machine is faster? By how much?

Let C be clock cycle.

Time(M1) = 2.125 × (8Y × C)
Time(M2) =

M1 is faster than M2 by

Example 4 (3/4)

c) To further improve the performance of the machines, a new compiler technique is introduced. The compiler can simply eliminate all class D instructions from the benchmark program without any side effects. (That is, there is no change to the number of class A, B and C instructions executed in the 2 machines.) With this new technique, which machine is faster? By how much?

Let Y = 1,000,000,000,000; Let C be clock cycle.

CPI(M1) = (3Y×1 + 2Y×2 + 2Y×3) / (3Y + 2Y + 2Y) = 13Y / 7Y = 1.857

CPI(M2) =

=

Time(M1) = 1.857 × (7Y × C)

Time(M2) =

M1 is faster than M2 by

Example 4 (4/4)

d) Alternatively, to further improve the performance of the machines, a new hardware technique is introduced. The hardware can simply execute all class D instructions in zero times without any side effects. (There is still execution for class D instructions.) With this new technique, which machine is faster? By how much?

Let Y = 1,000,000,000,000; Let C be clock cycle.

CPI(M1) = (3Y×1 + 2Y×2 + 2Y×3 + Y×0) / (3Y + 2Y + 2Y + Y)
= 13Y / 8Y = 1.625

CPI(M2) =
=

Time(M1) = 1.625 × (8Y × C)

Time(M2) =

M1 is faster than M2 by

Aspects of CPU Performance (1/2)

- Performance is determined by execution time.
- Does any of the following variables equal performance?
  - Number of cycles to execute a program?
  - Number of instructions in a program?
  - Number of cycles per second (cycle time)?
  - Average number of cycles per instruction?
  - Average number of instructions per second?
- Answer: No to all.
  - Common pitfall: thinking that one of the variables is indicative of performance when it really isn't.

Aspects of CPU Performance (2/2)

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

- CPU performance depends on:
  - Clock cycle time → Hardware technology and organisation
  - CPI → Organisation and ISA
  - Instruction count → ISA and compiler
- Be careful of the following concepts:
  - Machine → ISA and hardware organisation
  - Machine → cycle time
  - ISA + hardware organisation → number of cycles for any instruction (this is not average CPI)
  - ISA + compiler + program → number of instructions executed
  - Therefore, ISA + Compiler + Program + Hardware organisation + Cycle time → Total CPU time.

Key Concepts

- Performance is specific to a particular program.
  - Total execution time is a consistent summary of performance.
- For a given architecture, performance increase comes from:
  - Increase in clock rate (without adverse CPI effects)
  - Improvement in processor organisation that lowers CPI
  - Compiler enhancement that lowers CPI and/or instruction count
- Pitfall: expecting improvement in one aspect of a machine's performance to affect the total performance.

Reading Assignment

- Evaluating Performance
  - Read up COD sections 4.1 – 4.3 (3rd edition)
  - Read up COD section 1.4 (4th edition)

Benchmarking

- Benchmarking: Choosing programs to evaluate performance
  - Measure the performance of a machine using a set of programs which will hopefully emulate the workload generated by the user's programs.
- Benchmarks: programs designed to measure performance.

Benchmarks

Pros		Cons
• representative	Actual Target Workload	• very specific • non-portable • difficult to run or measure • hard to identify cause
• portable • widely used • improvements useful in reality	Full Application Benchmarks	• less representative
• easy to run, early in design cycle	Small "Kernel" Benchmarks	• easy to "fool"
• identify peak capability and potential bottlenecks	Microbenchmarks	• "peak" may be a long way from application performance

SPEC '95 (1/4)

- SPEC (Systems Performance Evaluation Cooperative)
  - Companies have agreed on a set of real program and inputs
  - 18 application benchmarks (with inputs) reflecting a technical computing workload
  - 8 integer
    - go, m88ksim, gcc, compress, li, ijpeg, perl, vortex
  - 10 floating-point intensive
    - tomcatv, swim, su2cor, hydro2d, mgrid, applu, turb3d, apsi, fppp, wave5
  - Must run with standard compiler flags
  - Eliminate special undocumented incantations that may not even generate working code for real programs
  - Can still be abused (Intel's "other" bug)
  - Valuable indicator of performance (and compiler technology)

SPEC '95 (2/4)

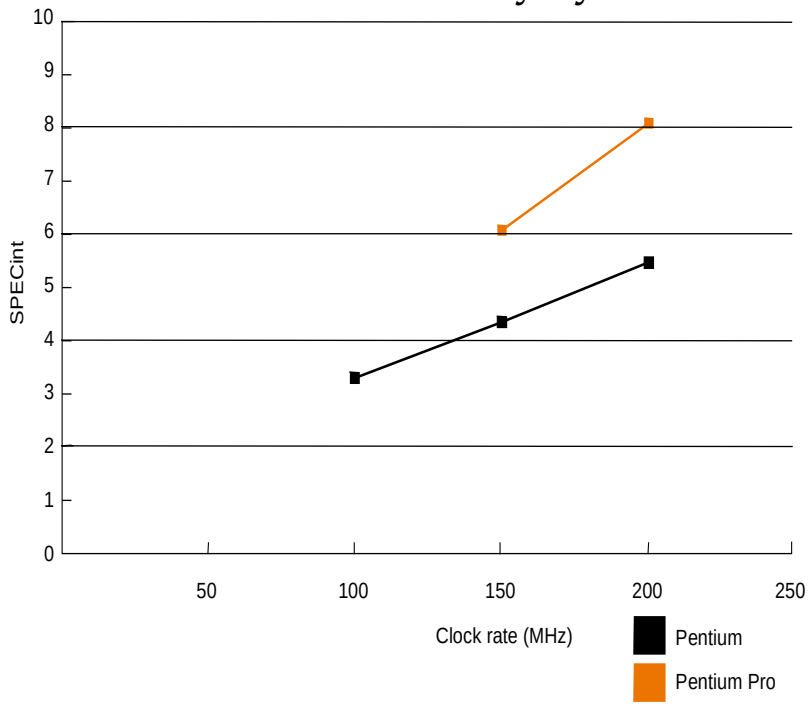
Benchmark	Description
go	Artificial intelligence; plays the game of Go
m88ksim	Motorola 88k chip simulator; runs test program
gcc	The Gnu C compiler generating SPARC code
compress	Compresses and decompresses file in memory
li	Lisp interpreter
ijpeg	Graphic compression and decompression
perl	Manipulates strings & prime numbers in the special-purpose prog. lang. Perl
vortex	A database program
tomcatv	A mesh generation program
swim	Shallow water model with 513 x 513 grid
su2cor	quantum physics; Monte Carlo simulation
hydro2d	Astrophysics; Hydrodynamic Naiver Stokes equations
mgrid	Multigrid solver in 3-D potential field
applu	Parabolic/elliptic partial differential equations
trub3d	Simulates isotropic, homogeneous turbulence in a cube
apsi	Solves problems regarding temperature, wind velocity, & distribution of pollutant
fpppp	Quantum chemistry
wave5	Plasma physics; electromagnetic particle simulation

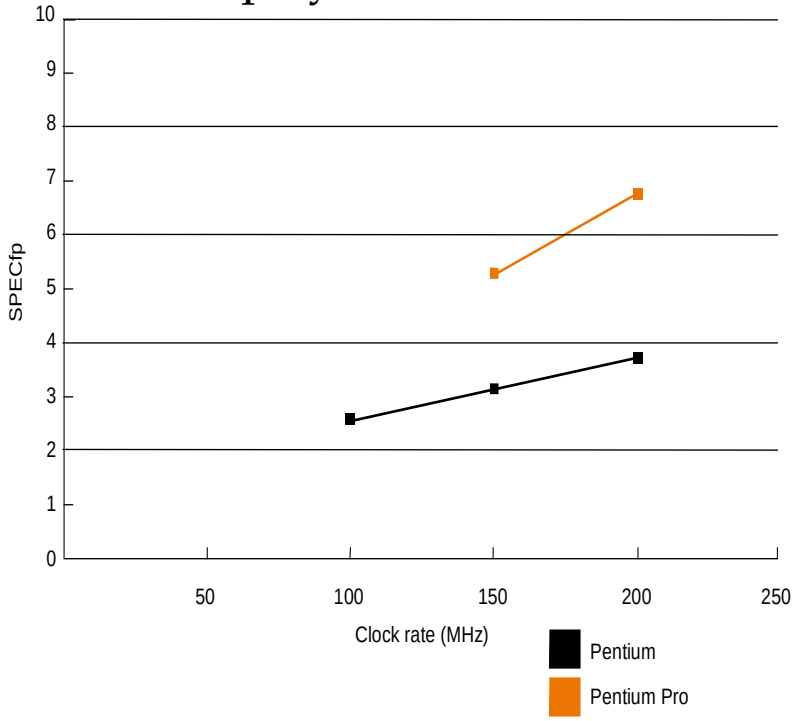
SPEC '95 (3/4)

- For a given ISA, increases in CPU performance can come from 3 sources:
  1. Increase in clock rate
  2. Improvements in processor organization that lower that CPI
  3. Compiler enhancements that lower the instruction count or generate instructions with a lower average CPI (e.g., by using simpler instructions)
- Next slide shows the SPECint95 and SPECfp95 measurements for a series of Intel Pentium processors and Pentium Pro processors.
  - Does doubling the clock rate double performance?
  - Can a machine with a slower clock rate have better performance?

SPEC '95 (4/4)

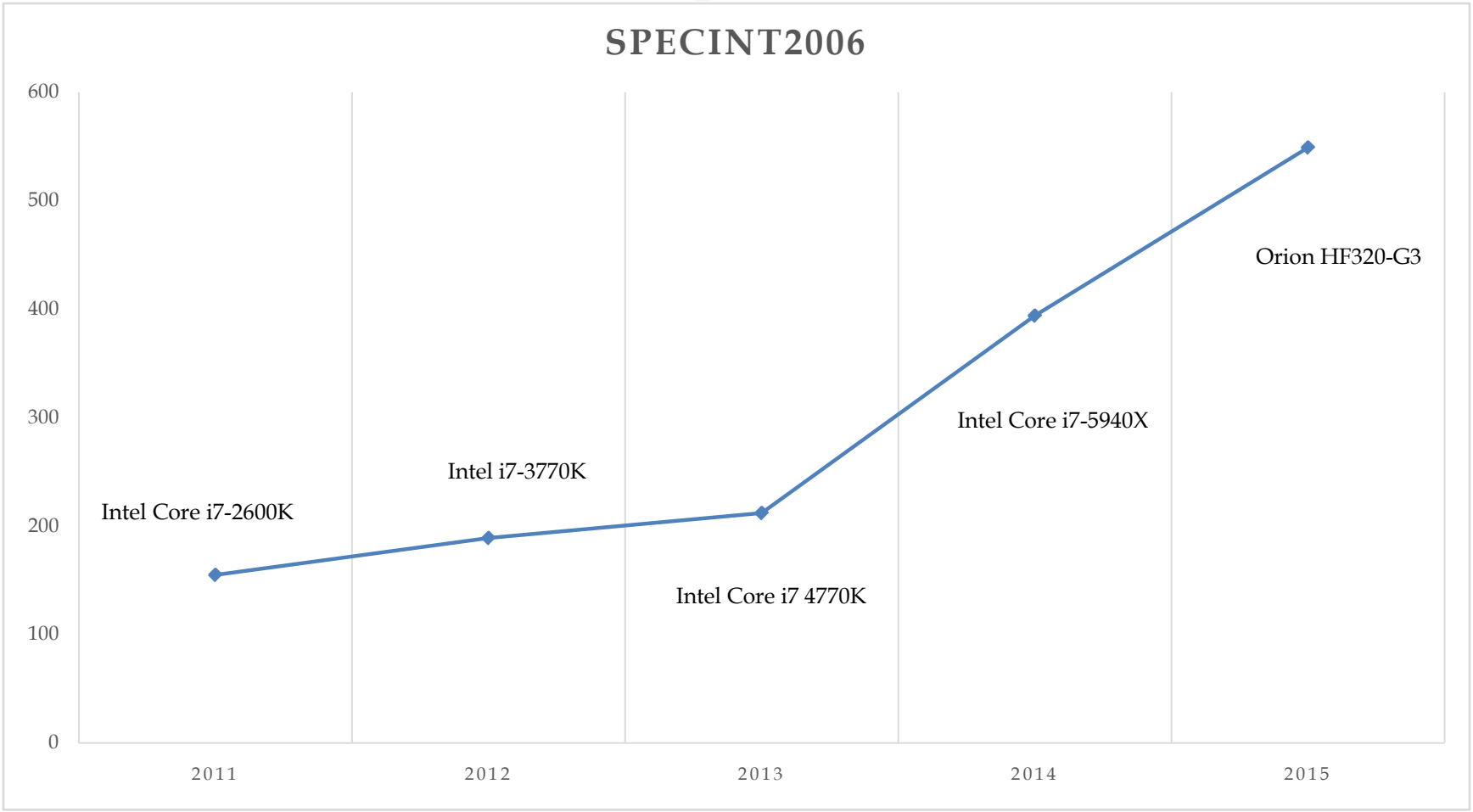
- At same clock rate, Pentium Pro is 1.4 to 1.5 times faster (for SPECint95) and 1.7 to 1.8 times faster (for SPECfp95) – improvements come from organizational enhancements (pipelining, memory system) to the Pentium Pro.
- Performance increases at a slower rate than increase in clock rate – bottleneck at memory system, Amdahl's law at play here.





Benchmark

KI angkatan 2015



Amdahl's Law (1/3)

- Pitfall: Expecting the improvement of one aspect of a machine to increase performance by an amount proportional to the size of the improvement.
- Example:
  - Suppose a program runs in 100 seconds on a machine, with multiply operations 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 4 times faster?

100 (total time) = 80 (for multiply) + UA (unaffected)
100/4 (new total time) =
➔Speedup =

Amdahl's Law (2/3)

- Example (continued):
  - How about making it 5 times faster?

100 (total time) = 80 (for multiply) + UA (unaffected)
100/5 (new total time) =
➔Speedup =

Amdahl's Law (3/3)

- This concept is the Amdahl's law. Performance is limited to the non-speedup portion of the program.
- Execution time after improvement = Execution time of unaffected part + (execution time of affected part / speedup)
- Corollary of Amdahl's law: Make the common case fast.

Example 5

- Suppose we enhance a machine making all floating-point instructions run **five times** faster. If the execution time of some benchmark before the floating-point enhancement is **12 seconds**, what will the speedup be if **half of the 12 seconds** is spent executing floating-point instructions?

Time =

Speedup =

Example 6

- We are looking for a benchmark to show off the **new floating-point unit** described in the previous example, and we want the overall benchmark to show a **speedup of 3**. One benchmark we are considering **runs for 100 seconds** with the old floating-point hardware. How much of the execution time would floating-point instructions have to account for in this program in order to yield our desired speedup on this benchmark?

Speedup =

Time_FI =

Reading Assignment

- SPEC Benchmarks
  - Read up COD sections 4.4 – 4.6 (3rd edition)
  - Read up COD sections 1.7 – 1.9 (4th edition)