Computer Architecture

Prof. Dr. Nizamettin AYDIN

naydin@yildiz.edu.tr
nizamettinaydin@gmail.com

http://www.yildiz.edu.tr/~naydin

1

Performance Metrics

2

Objectives

- How can we meaningfully measure and compare computer performance?
- Understand why program performance varies
  - Understand how applications and the compiler impact performance
  - Understand how CPU impacts performance
  - What trade-offs are involved in designing a CPU?
- Purchasing perspective vs design perspective

3

Outline

- Latency, delay, time
- Throughput
- Cost
- Power
- Energy
- Reliability

4

Basic Performance Metrics

- Latency, delay, time
  - Lower is better
    - Complete a task as soon as possible
  - Measured in sec, μs, ns
- Throughput (bandwith)
  - Higher is better
    - Complete as many tasks per time as possible
  - Measured in bytes/sec, instructions/sec
- Cost
  - Lower is better
    - Complete tasks for as little money as possible
  - Measured in $, TL, etc.

5

Basic Performance Metrics

- Power
  - Lower is better
    - Complete tasks while dissipating as few joules/sec as possible
- Energy
  - Lower is better
    - Complete tasks using as few joules as possible
  - Measured in Joules, Joules/instruction
- Reliability
  - Higher is better
    - Complete tasks with low probability of failure
  - Measured in Mean time to failure (MTTF)
    - MTTF: the average time until a failure occurs

6

Latency vs Throughput

Plane	Istanbul to Madrid (hours)	Speed	Passengers	Throughput Passenger/Hour
Aircraft 1	4 hrs	900 km /hr	400	100
Aircraft 2	4.8 hrs	750 km/hr	600	125

- Madrid to Istanbul is about 3600 km
- Time:
  - Aircraft 1 is faster than Aircraft 2
    - 900/750 = 1.2 times or 20% faster
- Throughput:
  - Aircraft 2 has a higher throughput
    - (750*600)/(900*400) = 1.25 times the throughput or 25% more throughput

7

Response Time vs Throughput

- Response time (latency)
  - the time between the start and the completion of a task
    - Important to individual users (passengers)
- Throughput (bandwidth)
  - the total amount of work done in a given time
  - Important to data center managers (airline)
- Different performance metrics are required
  - to benchmark embedded and desktop computers,
    - which are more focused on response time,
  - to benchmark servers,
    - which are more focused on throughput

8

Defining (Speed) Performance

- Minimizing the execution time maximizes the performance:

  performance of X = 1 / execution_time of X

- If X is n times faster than Y,
  - then the performance ratio n is

$$\frac{\text{performance of X}}{\text{performance of Y}} = \frac{\text{execution\_time of Y}}{\text{execution\_time of X}} = n$$

9

A Relative Performance Example

- If computer A runs a program in 10 seconds and computer B runs the same program in 15 seconds,
  - Which computer is faster?
  - How much faster?
- We know that A is n times faster than B if

$$\frac{\text{performance of A}}{\text{performance of B}} = \frac{\text{execution\_time of B}}{\text{execution\_time of A}} = n$$

- The performance ratio n is 15/10 =1.5
- So A is 1.5 times (50%) faster than B

10

Ratios of Measure: Side Note

- For bigger-is-better metrics,
  - improved means increase
    - $V_{new} = 2.5 * V_{old}$
      - A metric increased by 2.5 times (sometimes written 2.5x)
      - A metric increased by 150% (x% increase == 0.01*x+1 times increase)
- For smaller-is-better metrics,
  - improved means decrease
    - e.g., Latency improved by 2x, means latency decreased by 2x (i.e., dropped by 50%)
    - e.g., Battery life worsened by 50%, means battery life decrease by 50%.

11

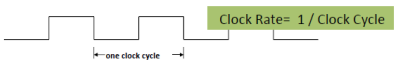
Examples

- Bigger-is-better examples
  - Bandwidth per dollar (e.g., in networking (GB/s)/$)
  - BW/Watt (e.g., in memory systems (GB/s)/W)
  - Work/Joule (e.g., instructions/joule)
  - In general
    - Multiply by big-is-better metrics, divide by smaller-is-better metrics
- Smaller-is-better examples
  - Cycles/Instruction (i.e., Time per work)
  - Latency * Energy -- Energy Delay Product
  - In general:
    - Multiply by smaller-is-better metrics, divide by bigger-is-better metrics

12

Clock Cycle and Clock Rate

- A clock cycle is a single electronic pulse of a CPU
  - To synchronize different parts of the circuit
  - To determine when events take place in the hardware
  - Processor runs at a constant clock rate
  - Clock cycle or tick or cycle = Discrete time interval
- Clock rate (frequency)
  - Number of clock cycles per second in hertz



- 1 nsec (10^{-9}) clock cycle => 1 GHz (10^{9}) clock rate
- 0.5 nsec clock cycle => 2 GHz clock rate

13

CPU Time (Execution Time)

- A program takes 15×10^{10} cycles to execute on a computer with a clock cycle time of 500 picosec.
  - How many seconds does it take for the program to execute?

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

- Clock Cycles:
  - How many cycles it takes for a program to execute!
- CPU Time (Execution time):
  - How many seconds it takes for a program to execute!

14

CPU Time Example

- Computer A has a 2GHz clock rate, executes a program in 10 sec (CPU time)
- Designing Computer B by aiming for 6 sec CPU time
  - With a faster clock, but this causes 1.2 × clock cycles
- What is Computer B's clock rate?

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

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

$$\text{Clock Rate}_B = \frac{1.2 \times 20 \times 10^9}{6s} = \frac{24 \times 10^9}{6s} = 4GHz$$

15

Clock Cycles per Instruction (CPI)

- Not all instructions take the same amount of time to execute
  - There is a mix of instructions in a program
    - E.g. Load, Store, ALU
  - Need to know the frequency of the instructions
    - Because instructions take different number of cycles to execute

Clock Cycles = Instruction Count × Cycles per Instruction

16

Instruction Mix and CPI

- All these values depend on the particular hardware implementation, not the ISA

Instruction Type	Cycles
Integer +, -, |, &, branches	1
Integer Multiply	3-5
Integer Divide	11-100
Floating Point +, -	3-5
Sqrt	7-27
Load and Stores	1-100s

- Values are for Intel's Nehalem processor

- Clock cycles per instruction (CPI)
  - the average number of clock cycles each instruction takes to execute
  - CPI is not the cycles required to execute a single instruction
  - A way to compare two different implementations of the same Instruction Set Architectures (ISA)

17

Comparing Computers

- Computers A and B implement the same ISA.
  - Computer A has a clock cycle time of 250 ps and an effective CPI of 2.0 for some program C
  - Computer B has a clock cycle time of 500 ps and an effective CPI of 1.2 for the same program.
- Which computer is faster and by how much?

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

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

18

Comparing Computers

Clock Cycles = Instruction Count×Cycles per Instruction
CPU Time = Instruction Count×CPI×Clock Cycle Time

- Each computer executes the same number of instructions, I, so
  CPU time$_A$ = I × 2.0 × 250 ps = 500 × I ps
  CPU time$_B$ = I × 1.2 × 500 ps = 600 × I ps
- Clearly, A is faster than B by the ratio of execution times

$$\frac{\text{performance}_A}{\text{performance}_B} = \frac{\text{execution\_time}_B}{\text{execution\_time}_A} = \frac{600 \times I \text{ ps}}{500 \times I \text{ ps}} = 1.2$$

19

CPU Performance

- Different programs do different amounts of work
  - e.g., Playing a DVD vs writing a word document
- The same program may do different amounts of work depending on its input
  - Compiling a 1000-line program vs compiling a 100-line program
- The same program may require a different number of instructions on different ISAs
  - MIPS vs. x86
- To make a meaningful comparison between two computer systems, they must be doing the same work.
  - They may execute a different number of instructions (e.g., because they use different ISAs or a different compilers)
  - But the task they accomplish should be exactly the same.

20

CPU Performance

CPU time = Instruction_count × CPI / Clock Rate

	Instruction_ count	CPI	Clock Rate
Algorithm	x	x	
Programming Language	x	x	
Compiler	x	x	
ISA	x	x	x

Clock Rate = 1 / Clock Cycle

21

Compiler Benefits

- Comparing performance for bubble sort
  - To sort 100,000 words with the array initialized to random values

optimizations levels on gcc	Relative performance	Clock cycles (M)	Instr count (M)	CPI
None	1.00	158,615	114,938	1.38
O1	2.37	66,990	37,470	1.79
O2	2.38	66,521	39,993	1.66
O3	2.41	65,747	44,993	1.46

  - The unoptimized code has the best CPI, the O1 version has the lowest instruction count, but the O3 version is the fastest.
- Instruction count and CPI are not good performance indicators in isolation
- Compiler optimizations are sensitive to the algorithm

22

Instruction Count

- Note that instruction count is dynamic
  - its not the number of lines in the code, or
  - number of lines in an assembly code that compiler generates
- Static instruction count refers to the program as it was compiled
- Dynamic instruction count refers to the program at runtime
- Dynamic instruction count is more accurate
  - For example, you have a loop in your program then some instructions get executed more than once or
  - In the presence of branches, some instructions may not be executed at all.

Type	CPI	Static #	Dyn#
mem	5	1	1
int	1	6	44
br	1	2	21
Total	1.06	9	66

- Average CPI:
  (5×1 + 1×44 + 1×21)/66= 1.06

23

Instruction Mix

- Measure MIPS instruction executions in benchmark programs (e.g. SPEC)
  - Consider making the common case fast
  - Consider compromises

Instruction class	MIPS examples	SPEC2006 Int	SPEC2006 FP
Arithmetic	`add, sub, addi`	16%	48%
Data transfer	`lw, sw, lb, lbu, lh, lhu, sb, lui`	35%	36%
Logical	`and, or, nor, andi, ori, sll, srl`	12%	4%
Cond. Branch	`beq, bne, slt, slti, sltiu`	34%	8%
Jump	`j, jr, jal`	2%	0%

24

Dynamic Frequency

- Most multi-core architectures nowadays support dynamic voltage and frequency scaling (DVFS) to adapt their speed to the system's load and save energy.
  - Enabled by the request from the Operating System
- A core can exceed the its manufactured operation frequency
  - Intel's Turbo Boost and AMD Turbo CORE
- Increased clock rate is limited by the power, current and thermal limits
  - This is not similar to hearth rate increase
  - CPU runs at a higher rate for awhile, it is discrete

25

26