# CS 305 Computer Architecture

## Measuring and Reporting Performance

Prof. Umesh Bellur

## Different Perspectives

- The ***User*** perspective
  - How fast is it?
  - How much does it cost in terms of money?
- The ***Designer*** perspective
  - Which components should I use to ensure a certain speed?
  - What does it cost in terms of components?
- The ***Operator*** Perspective
  - Do I have the controls to tradeoff speed with cost?
  - What is the cost of operation?
- All of the perspectives need quantitative definitions of what performance is.

# An Analogy from the Airline Business

| Airplane | Passenger capacity | Cruising range (miles) | Cruising speed (m.p.h) | Passenger throughput (Passenger × m.p.h) |
|---|---|---|---|---|
| Boeing 777 | 375 | 4630 | 610 | 228,750 |
| Boeing 747 | 470 | 4150 | 610 | 286,700 |
| BAC/Sud Concorde | 132 | 4000 | 1350 | 178,200 |
| Douglas DC-8-50 | 146 | 8720 | 544 | 79,424 |

How to measure performance for an airplane?

Cruising speed (How fast it gets to the destination)
Flight range (How far it can reach)
Passenger capacity (How many passengers it can carry)

**Derived Measure: Passenger Throughput**

Such derived measures normalize across different selections!

# Measuring and reporting computer performance

- What does it mean to say one computer is faster than another?
- ***User***: the faster one runs a program in less time
  - Response time – time between start and completion of a task
- ***Data center manager***: the faster one completes more jobs or queries per unit time
  - Throughput – total work done in a given time.
- The common aspect is TIME: the hardware or software that performs the same amount of work in less time is the fastest
- What is important depends on the type of the device:
  - For a mobile phone or a laptop – response time is important
  - For a server – throughput is important.

# Comparing alternatives

- The phrase "A is faster than B" means that the response time or execution time is lower on A than on B for the given task
- "A is ***n*** times faster than B" means

$$\frac{\text{Execution time}_B}{\text{Execution time}_A} = n = \frac{\text{Performance}_A}{\text{Performance}_B}$$

- Increasing performance decreases execution time
- To avoid confusion with "increasing" and "decreasing" use "improve performance" or "improve execution time"

# How to measure Execution time?

- Wall-clock time, response time, or elapsed time: the latency to complete a task including disk accesses, memory accesses, I/O activities, O/S overhead – everything
- UNIX `time` command example output

```
        90.7u 12.9s 2:39 65%
User CPU time = 90.7 seconds
O/S CPU (Sys) time = 12.9 seconds
Elapsed time = 2 minutes 39 seconds
% elapsed time that is CPU time = 65
```

(More than 1/3 of elapsed time was waiting for I/O or running other programs or both)

# CPU Exec Time

**# of Clock Cycles x Period of one cycle**

**= # of clock cycles / clock frequency**

- Thus we can reduce execution time by either reducing the number of cycles OR by increasing the frequency
- However these are not independent parameters
  - When I increase the clock frequency the period reduces and thus I may need more cycles per instruction.

# Example

*A program runs in **10 seconds on a computer "A" with a 400 MHz clock**.*
*We desire a faster computer **"B"** that could run the program **in 6 seconds**.*
*The designer has determined that a substantial increase in the clock speed is possible, however it would cause computer **"B" to require 1.2 times as many clock cycles as computer "A"**. What should be the clock rate of computer "B"?*

$$\text{CPU time (A)} = \frac{\text{CPU clock cycles}}{\text{Clock freq (A)}} \qquad 10 \text{ seconds} = \frac{\text{CPU clock cycles of program}}{400 \times 10^6 \text{ cycles/second}}$$

$$\text{CPU clock cycles of program} = 10 \text{ seconds} \times 400 \times 10^6 \text{ cycles/second} = 4000 \times 10^6 \text{ cycles}$$

To get the clock freq of the faster computer, we use the same formula

$$6 \text{ seconds} = \frac{1.2 \times \text{CPU clock cycles of program}}{\text{clock rate (B)}} = \frac{1.2 \times 4000 \times 10^6 \text{ cycles}}{\text{clock rate (B)}}$$

$$\text{clock rate (B)} = \frac{1.2 \times 4000 \times 10^6 \text{ cycles}}{6 \text{ second}} = 800 \times 10^6 \text{ cycles/second}$$

8

# From an Instruction perspective

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

$$\text{CPU time} = \text{Instruction count} \times \text{CPI} \times \text{Clock cycle time}$$

Or

$$\text{CPU time} = \frac{\text{Instruction count} \times \text{CPI}}{\text{Clock freq}}$$

| Component of performance | Units of measure |
|---|---|
| CPU execution time for a program | Seconds for the program |
| Instruction count | Instructions executed for the program |
| Clock cycles per instructions (CPI) | Average number of clock cycles/instruction |
| Clock cycle time | Seconds per clock cycle |

9

# Example

*Suppose we have two implementation of the same instruction set architecture.* ***Machine "A" has a clock cycle time of 1 ns and a CPI of 2.0 for some program, and machine "B" has a clock cycle time of 2 ns and a CPI of 1.2 for the same program****. Which machine is faster for this program and by how much?*

Both machines execute the same instructions for the program. Assume the number of instructions is "I",

CPU clock cycles (A) = I × 2.0 CPU clock cycles (B) = I × 1.2

The CPU time required for each machine is as follows:

CPU time (A) = CPU clock cycles (A) × Clock cycle time (A)
= I × 2.0 × 1 ns = 2 × I ns

CPU time (B) = CPU clock cycles (B) × Clock cycle time (B)
= I × 1.2 × 2 ns = 2.4 × I ns

Therefore machine A will be faster by the following ratio:

$$\frac{\text{CPU Performance (A)}}{\text{CPU Performance (B)}} = \frac{\text{CPU time (B)}}{\text{CPU time (A)}} = \frac{2.4 \times I \text{ ns}}{2 \times I \text{ ns}} = 1.2$$

10

# Instruction Classes

*A compiler designer is trying to decide between two code sequences for a particular machine. The hardware designers have supplied the following facts:*

| Instruction class | CPI for this instruction class |
|---|---|
| A | 1 |
| B | 2 |
| C | 3 |

*For a particular high-level language statement, the compiler writer is considering two code sequences that require the following instruction counts:*

| Code sequence | Instruction count for instruction class | | |
|---|---|---|---|
| | A | B | C |
| 1 | 2 | 1 | 2 |
| 2 | 4 | 1 | 1 |

*Which code sequence executes the most instructions? Which will be faster? What is the CPI for each sequence?*

**Answer:**

Sequence 1: executes 2 + 1 + 2 = 5 instructions

Sequence 2: executes 4 + 1 + 1 = 6 instructions ✓

11

# Comparing Code Segments

Using the formula: $$\text{CPU clock cycles} = \sum_{i=1}^{n} CPI_i \times C_i$$

Sequence 1: CPU clock cycles = (2 ×1) + (1 ×2) + (2 ×3) = 10 cycles

Sequence 2: CPU clock cycles = (4 ×1) + (1 ×2) + (1 ×3) = 9 cycles

☞ Therefore Sequence 2 is faster although it executes more instructions

Using the formula: $$\text{CPI} = \frac{\text{CPU clock cycles}}{\text{Instruction count}}$$

Sequence 1: CPI = 10/5 = 2

Sequence 2: CPI = 9/6 = 1.5

☞ Since Sequence 2 takes fewer overall clock cycles but has more instructions it must have a lower CPI

12

# Performance of a program

- Depends on:

1. Algorithm – determines the number of instructions
2. PL and Compiler – also determines the number of instructions
3. ISA – determines CPI and max clock rate.

| | Instr. Count | CPI | Clock Rate |
|---|---|---|---|
| **Program** | X | | |
| **Compiler** | X | X | |
| **Instruction Set** | X | X | |
| **Organization** | | X | X |
| **Technology** | | | X |

# CPU Utilization and Load

- aka % CPU
- Reflects the fraction of a time slice the CPU is busy.
- Also indicative of "load" on a computer
- Sometimes "load average" is also indicative of the average number of processes waiting for the CPU averaged across all time slices.

```
> uptime
  12:28  up 63 days, 23:52,
  4 users,
  load averages: 1.32 1.33 1.45
```

## Fundamental principle of performance

- Make the common case fast
  - In making a design decision, favor the frequent case over the infrequent case
  - Example: overflow is rare so optimize the ALU circuit for the non-overflow case
- Challenge: determine what is the frequent case and how much performance can be improved by making it faster

## Paths to Upgrading a computer

1. Get faster processors (higher clock speed)
2. Get more processors

What is the effect of each of these on RT and Throughput?

- Faster processors lowers the response time and therefore increase the throughput
- More processors keep the response time the same while increasing throughput – is this really true? Can response time decrease while adding more processors for a single threaded application?

# The Big Change

- From Uni processors to Multi processors
  - Driven by heat dissipation and density of the circuit considerations.
- However it's hard to write correct parallel programs.
  - Synchronization and communication often negates the gains gotten by parallelization of the program.

# Speedup

$$\text{Speedup} = \frac{\text{Performance for entire task using the enhancement when possible}}{\text{Performance for entire task without using the enhancement}}$$

or, reformulating

$$\text{Speedup} = \frac{\text{Execution time for entire task without using the enhancement}}{\text{Execution time for entire task using the enhancement when possible}}$$

# Amdahl's Law - Speedup from enhancement

Depends on two factors:

1. The fraction of the computation time that the original system can be converted to take advantage of the enhancement.
   **$F_e$ = Fraction$_{enhanced}$ <= 1**

2. The improvement gained by the enhanced mode, that is, how much faster is the enhanced mode than the original mode.
   **$S_e$ = Speedup$_{enhanced}$ > 1**

# Amdahl's Law

$$\text{Speedup}_{overall} = \frac{\text{Execution time}_{original}}{\text{Execution time}_{enhanced}}$$

$$= \frac{1}{(1 - F_e) + (F_e / S_e)}$$

Fraction unchanged

Fraction sped up

# Examples

- An enhancement runs 10 times faster than the original and can be used 40% of the time. What is the overall speedup?

$$F_e = 0.4$$

$$S_e = 10$$

$$S_{overall} = \frac{1}{(1 - 0.4) + 0.4/10}$$

$$= 1/0.64$$

$$= 1.56$$

- Implementations of FP square root (FPSQR) vary widely in performance. Suppose FPSQR is 20% of the execution time in a critical task and FP overall is 50% of the total time. Fix #1: add FPSQR hardware to speed up 10 times. Fix #2: speed up all FP operations by 2 times. Assume equal "cost" for either fix. Which is better?

- $S_{FPSQR} = \frac{1}{(1 - 0.2) + 0.2/10} = \frac{1}{0.82} = 1.22$

- $S_{FP} = \frac{1}{(1 - 0.5) + 0.5/2} = \frac{1}{0.75} = 1.33$ ← **better speedup**

# A "limiting" example

- Execution time = 100s
- Multiply operations make up 80% of the code.
- Suppose I want the new execution time to be 20 s (or a 5x speedup) – how fast should I make the multiplication?

```
Assume the multiply op becomes x times as
fast
Exec time after improvement = 80/x + 20
=> 20 = 80/x + 20 (new exec time is to be 20s)
=> x = ∞
```

In other words, its impossible to make this happen.

# Other uses of Amdahl's Law: Dependability

- Computers are designed and built at different levels of abstraction
  - E.g., start with RPi and descend until you reach the individual transistors
- Some faults are widespread, e.g., power failure
- Some faults are limited to a single component in a module
- Utter failure of a module at one level of abstraction may be viewed as a component failure from a higher level

24

## When is a system operating properly?

- Systems alternate between two SLA states
  - *(1) Service functioning*
  - *(2) Service interruption*
- Transition (1) to (2) caused by *failure*, from (2) to (1) by *restoration*
- Quantifying these transitions gives two main measures of Dependability

25

## Module *Reliability* and *Availability*

- Module reliability measures continuous service functioning from a reference initial instant (also termed *time to failure*)
  - MTTF (mean time to failure) is a **reliability** measure
  - MTTF units typically # failures per 10^9 hours
    - 1,000,000 hr MTTF means 1 failure in time 10^6 hours
    - Or 1000 failures in time 10^9 hours
  - Service interruption measured in *Mean Time to Repair* (MTTR) in hours
  - Often quoted for a module is mean time between failures, MTBF = MTTF+*MTTR*, but MTTF should be used

26

# Module *Availability*

- Availability measures service accomplishment with respect to alternation between state of accomplishment and interruption
- **Module availability** = $\frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}}$
- Reliability and availability are quantifiable, rather than synonyms for dependability

27

# Amdahl's Law MTTF example

- Assume a disk subsystem of a computer with
  - 10 disks each rated at 1,000,000-hour MTTF
  - 1 ATA disk controller, 500,000-hour MTTF
  - 1 power supply, 200,000-hour MTTF
  - 1 cooling fan, 200,000-hour MTTF
  - 1 ATA cable, 1,000,000-hour MTTF
- Using simplifying assumptions, so MTTFs can be added
  - Module lifetimes are exponentially distributed (means that probability of module failure is not a function of module age)
  - Module failures are independent

28

## Coping with failure: Redundancy

- Redundancy in time (Try, try again principle)
  - Repeat the operation to see if it still produces an error
- Redundancy in resources (physical space)
  - Have component(s) standing by to take over from one that failed
  - Have multiple components producing results in parallel plus a disagreement resolution scheme
- Once a failed component is "replaced," system is assumed to be as good as new

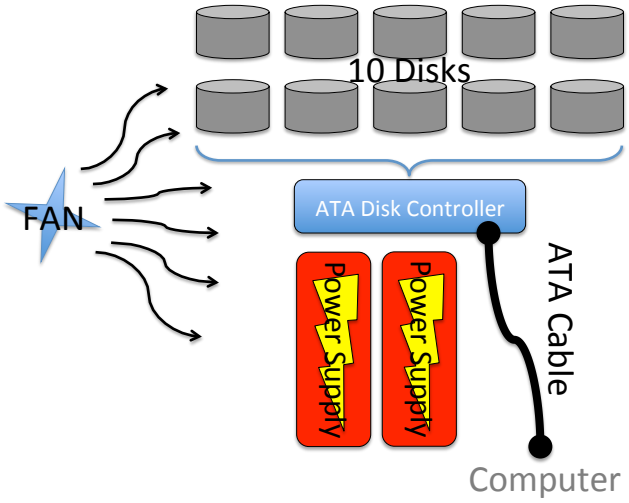
29

## Dependability Rule

- Design so no single component failure can cause system failure

30

# Disk subsystem w/ second supply




31

# Quantify dual power supply reliability

- Mean time until one power supply fails is
  ½ $MTTF_{power\ supply}$
- "1/2" factor illustrates TANSTAAFL Principle
  - TANSTAAFL = There Ain't No Such Thing As A Free Lunch
  - TANSTAFFL says there is always a cost for what you get, even when it may look like what you get is free
  - Origin: Bars would offer a "free lunch" to draw in customers, but lunch was very salty food and the price of the beers high
  - Example today: Google searches, free information in exchange for tracking your activity and targeting you with ads
- Good approximation of probability of second failure taking the system down is MTTR/MTTF
  - MTTR viewed as duration of the "window of vulnerability" to a single fault, given 1 supply is failed

32

## Put it all together

- $MTTF_{power\ supply\ pair} = MTTF^2 / (2 \times MTTR)$
- Assume is takes 24 hours for human operator to notice a power supply has failed and to replace it
- $MTTF_{pair} = 200{,}000^2/2x24 = 830{,}000{,}000$ hours versus $MTTF_{single\ power\ supply}$= 200,000 hours
- Power supply reliability improved by 4,150x

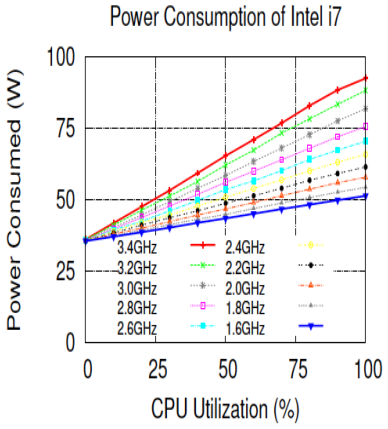
33

## Using Amdahl's Law

- Failure rate of original single power supply was 5 per million hours out of 23 per million hours for the whole system
- Dual power supply improves failure rate by 5/23=0.22 = $F_e$ and factor of 4150 = $S_e$
- Amdahl's Law says improvement $1/(1-F_e+ F_e/S_e)$
- Improvement = 1/(1-0.22+0.22/4150) = 1.28
- Impressive 4150x improvement in power supply reliability improves system reliability by a noticeable but small 28%
- Fan is now the least reliable component and a single point of failure (violates reliability rule)

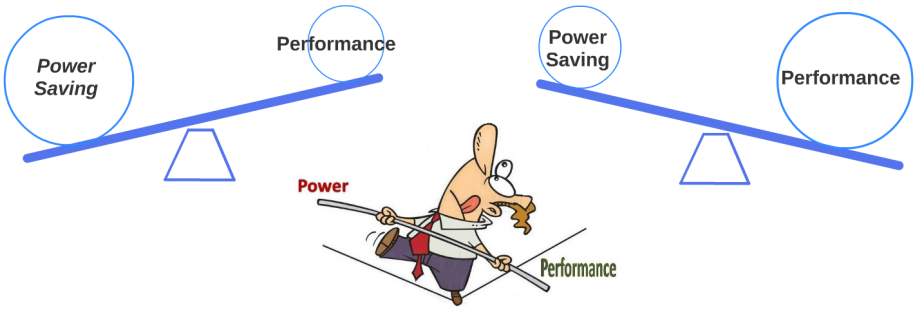
34

# The new face of architecture




- Modern processors operate at around 35% of the peak power when idle (as opposed to 65% for older processors).
  - Some of this is leakage at the transistor level.
- Multiple packages each of which has multiple cores (Xeon 8 core in two 4-core packages)
- They also offer Dynamic Voltage and Frequency scaling knobs

# The power performance equation




We can also tradeoff amount of CPU for frequency to minimize power consumption