Computer Architecture. Week 2

Muhammad Fahim, Alexander Tormasov

Innopolis University

m.fahim@innopolis.ru
a.tormasov@innopolis.ru

September 10, 2020

Topic of the lecture

- Performance Metrics of a System

Topic of the tutorial

- Basics of Combinational Logic

Topic of the lab

- Quartus Prime
- ModelSim

Content of the Class

- Review of the evolution of the speed of computer systems
- The role of performances
- Measuring performances
- How to combine different performance measures to make decisions
- Programs to determine comprehensive performance indexes

Evolution of the Speed

- Rapidly changing field:
  - Vacuum tube $\rightarrow$ transistor $\rightarrow$ IC $\rightarrow$ VLSI
  - Doubling every 1.5 years:
    - Memory capacity
    - Processor speed

Increase of Technology Enabled Abstraction

- Delving into the depths reveals more information
- An abstraction omits unneeded detail, helps us cope with complexity

Increase of Technology Enabled Abstraction

- High-level Language Program

```
swap(int v[], int k){
int temp;
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
}
```

- Assembly Language Program (for MIPS)

```
swap:
    multi $2, $5,4
    add   $2, $4,$2
    lw    $15, 0($2)
    lw    $16, 4($2)
    sw    $16, 0($2)
    sw    $15, 4($2)
    jr    $31
```

- Binary machine language program (for MIPS)

```
00000000101000100000000100011000
00000000100000100001000000100001
10001101111100010000000000000000
10001111000010010000000000000100
10101111000010010000000000000000
10101101111100010000000000000100
00000011111000000000000000001000
```

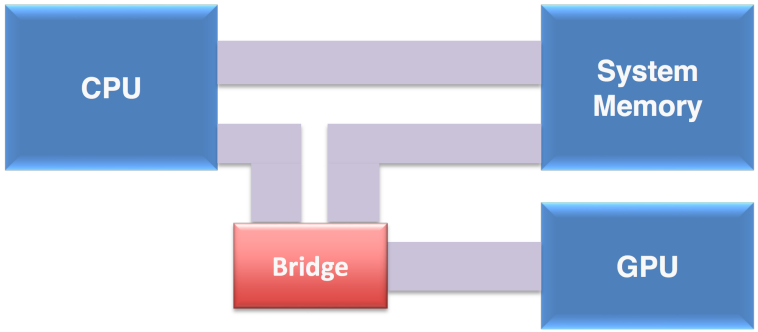
Instruction Set Architecture (ISA)

A very important abstraction

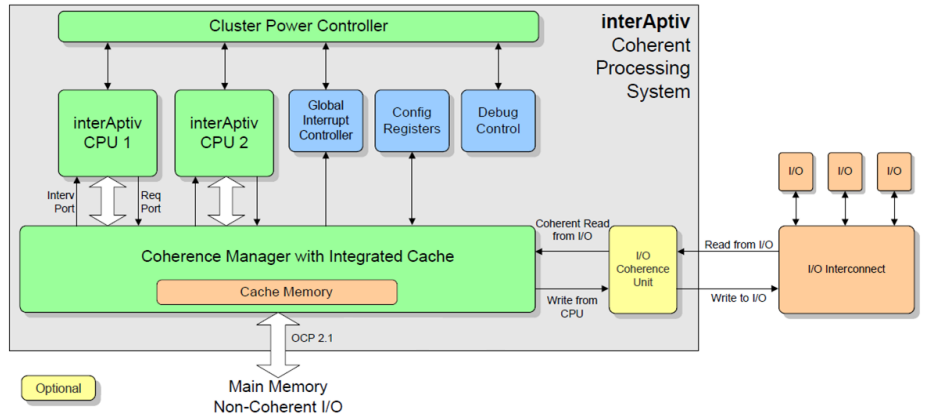
- Interface between hardware and low-level software
- Standardizes instructions, machine language bit patterns, etc.
- **Advantage**: different implementations of the same architecture
- **Disadvantage**: sometimes prevents using new innovations

Organization of Computers

Simplified organization (without Cache)



Complex organization



Note: Real-life situation can be more complex

Source: Imagination Community Program

Organization of Computers

- Each computing system consists of the following types of components:
  - Computational devices: one or many CPU cores, GPU, DSP[1], SMP[2]
  - Memory hierarchy: hardware caches, NUMA[3], memory controllers
  - I/O devices with memory-mapped registers
  - Interconnect bus fabric that connects all computational and I/O devices to the system memory via regular read and write requests

[1] Digital Signal Processor
[2] Symmetric MultiProcessing
[3] Non-Uniform Memory Access

Simplified View of a Computers – 1

- For simplicity, textbook uses the terms processor/CPU to refer:
  - Datapath
  - Control
  - Memory
  - Input devices
  - Output devices
- In the past this corresponded to the actual structure of a processor, but now the structure of a processor has evolved in the lines mentioned previously in slide 11.
- So, overall we will use in the explanation the terms above just for consistency and backward compatibility with the terms computer scientists and general people typically use.

Simplified View of a Computers – 2

- Not all "memory" is created equally
  - Hardware Cache: fast (expensive) memory are placed closer to the processor
  - Main memory: less expensive memory–we can have more
  - Secondary memory: even less expensive and we can have much more
- Input and Output (I/O) devices have the messiest organization
  - Wide range of speed: graphics vs. keyboard
  - Wide range of requirements: speed, standard, cost
  - Least amount of research (so far)

Content of the Class

- Review of the evolution of the speed of computer systems
- The role of performances
- Measuring performances
- How to combine different performance measures to make decisions
- Programs to determine comprehensive performance indexes

Performance – why?

- Measure, report, and summarize
- Make intelligent choices
- See through the marketing hype
- Key to understand the underlying organizational motivation

Typical Questions

- Why some hardware is better than others for different programs?
- What factors of system performance are hardware related?
  - For example: Do we need a new machine, or a new operating system?
- How does the machine's instruction set affect performance?

Best Performance

Which of these airplanes has the best performance?

Airplane	**Passengers**	**Range(miles)**	**Speed(mph)**
Boeing 737-100	101	630	598
Boeing 747	470	4150	610
BAC/Sud Concorde	132	4000	1350
Douglas DC-8-50	146	8720	544

- How much faster is the Concorde compared to the 747?
- How much bigger is the 747 than the Douglas DC-8?

What if we consider a new index?

- Passenger Throughput (PT)

Airplane	Passengers	Range (miles)	Speed (mph)	PT (pass*mph)
Boeing 737-100	101	630	598	60398
Boeing 747	470	4150	610	286700
BAC/Sud Concorde	132	4000	1350	178200
Douglas DC-8-50	146	8720	544	79424

Content of the Class

- Review of the evolution of the speed of computer systems
- The role of performances
- Measuring performances
- How to combine different performance measures to make decisions
- Programs to determine comprehensive performance indexes

How do we Define Performances?

The GQM

- Goal – set the goal why you measure
- Question – set suitable questions you are interested in determining
- Metrics – set up a suitable measurement device

- Your example 1: performances of a car
- Your example 2: performances of a computer system

GQM Source: Caldiera, V. R. B. G., and H. Dieter Rombach. "The goal question metric approach." Encyclopedia of software engineering 2.1994 (1994): 528-532.

Performances of a Computer System

- Goal
  - To measure how fast is a computer system
- Questions
  - 1. How long does it take for a job to run?
  - 2. How many jobs machine can run at once?

Two Kinds of Indexes

- How long does it take for a job to run?
  - Latency (Response time)
- How many jobs machine can run at once?
  - Throughput

Latency (Response Time)

Also known as execution time

- How long does it take for my job to run?
- How long does it take to execute a job?
- How long must I wait for the database query?

Different Execution Time

- Elapsed Time
  - Counts everything (disk and memory accesses, I/O , etc.)
  - A useful number, but often not good for comparison purposes

- CPU time
  - Doesn't count 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 that are "in" our program

Throughput

- How many jobs can the machine run at once?
- What is the average execution rate?
- How much work is getting done?

Response Time and Throughput

- Response time and throughput are related . . . sometimes
  - Replacing a processor with a faster yields both improves of response time and throughput
  - Adding a second processor, improves the throughput but NOT the response time

Latency vs Throughput (Sports car vs Bus)

- Imagine a group of people who want to go from Innopolis to Kazan. If we use a sports car to transport one person at a time, individual people get there faster.
- Hence, the latency is high. But to transport the entire group of people it takes much longer; the sports car needs to make n trips(i.e., the throughput of the system is low).
- Now if we had a bus, we could have transported all of those people together in one go. In this case, the throughput would have been high, but the latency, which describes the speed per person, will go down.
- We cannot say what is better, having a higher latency or having a higher throughput. It is implementation dependent. Different applications might require different criteria.

Performances

- For some program running on a machine X

$$Performance_X = \frac{1}{Executiontime_X}$$

- X is n times faster than Y

$$\frac{Performance_X}{Performance_Y} = n$$

- Problem: A runs a program in 20 seconds; B runs the same program in 25 seconds; how faster is A than B?

$$\frac{Performance_A}{Performance_B} = \left(\frac{1}{Executiontime_A}\right) / \left(\frac{1}{Executiontime_B}\right)$$

$$= \frac{Executiontime_B}{Executiontime_A} = \frac{25}{20} = 1.25$$

The Notion of Clock Cycle

- Inside computers there is a device which measures time, the clock
- The clock determines the sequencing of events ...
- ... via clock cycles of constant time, also known as clock periods, clock ticks, or just ticks
- The duration is called cycle time
- The clock rate is $(\text{tick})^{-1}$
- Clock rate is also known as clock frequency.
- Clock generator: Generates the signals
- The clock signal is measured in Hertz (Hz)

Content of the Class

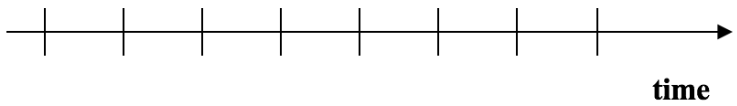
- Review of the evolution of the speed of computer systems
- The role of performances
- Measuring performances
- How to combine different performance measures to make decisions
- Programs to determine comprehensive performance indexes

Using Clock Cycles

- Instead of reporting execution time in seconds, we often use cycles

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

- Clock ticks indicate when to start activities (one abstraction)



The Clock Rate

We have often heard the concept of clock rate:

- Clock rate (frequency) = cycles per second (1 Hz. = 1 cycle/sec)
- For Example: 200 MHz. clock
  - It executes 200 millions cycles per second
  - It has a cycle time:

$$\frac{1}{200 \times 10^6} = \frac{1}{2} \times 10^{-8} = 5 \times 10^{-9}\ s = 5\ ns$$

CPU Performance

- For a Program (Program is shortly written as PGM):

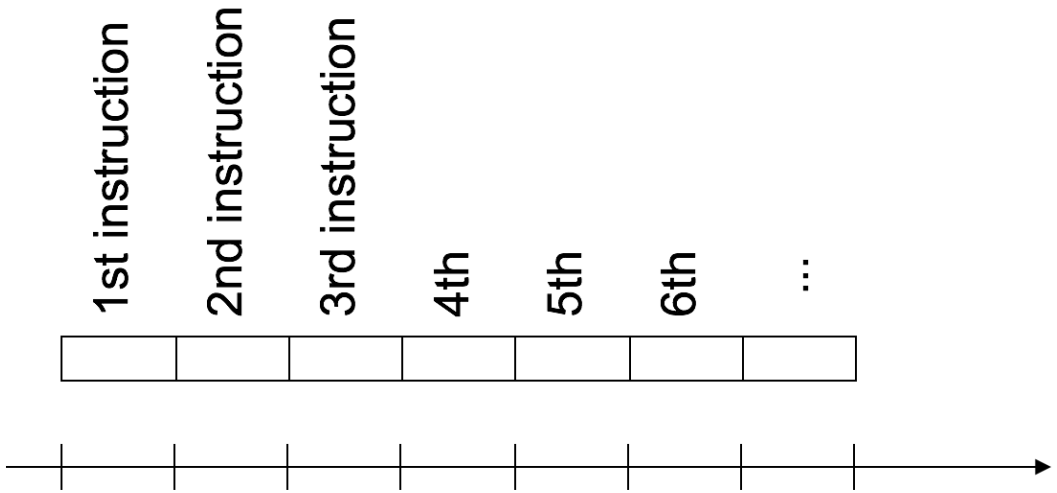
$$ExecutionTime(PGM) = ClockCycles(PGM) \times CycleTime$$

- Alternatively, because clock rate and clock cycle time are inverses

$$ExecutionTime(PGM) = \frac{ClockCycles(PGM)}{ClockRate}$$

How many Cycles are Required for a Program?

- Can we assume that # of cycles = # of instructions?



Solution

- This assumption is incorrect
- Different instructions take different amounts of time on different machines
- Why? Remember that these are machine instructions, not lines of Java code

Different # of Cycles for Different Instructions

- Multiplication takes more time than addition
- Floating point operations take longer than integer ones
- Accessing memory takes more time than accessing registers
- In pipeline processors multiple instruction may be executed at the same time.

- **Important point:** changing the cycle time often changes the number of cycles required for various instructions (more later)

Now that we Understand Cycles

- A given program will require
  - Some number of instructions (machine instructions)
  - Some number of cycles
  - Some number of seconds
- We have a vocabulary that relates these quantities:
  - Cycle time (seconds per cycle)
  - Clock rate (cycles per second)
  - CPI (average cycles per instruction) this is an average!!
  - MIPS (millions of instructions per second), higher for programs using simple instruction sets

The Classic CPU Performance Equation

$$CPU\ time = Instruction\ count \times CPI \times clock\ cycle\ time$$

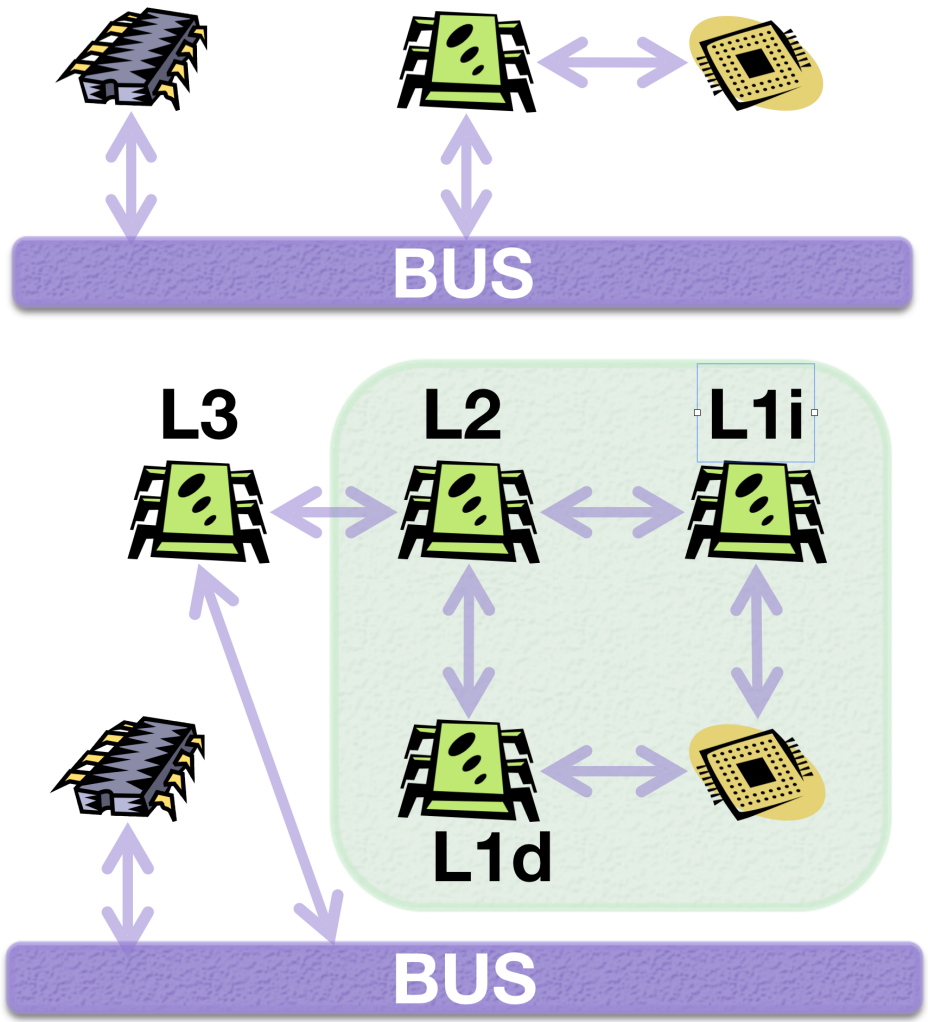
Since the clock rate is the inverse of clock cycle time:

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

NOTE:
Instruction count: the number of instructions executed by the program

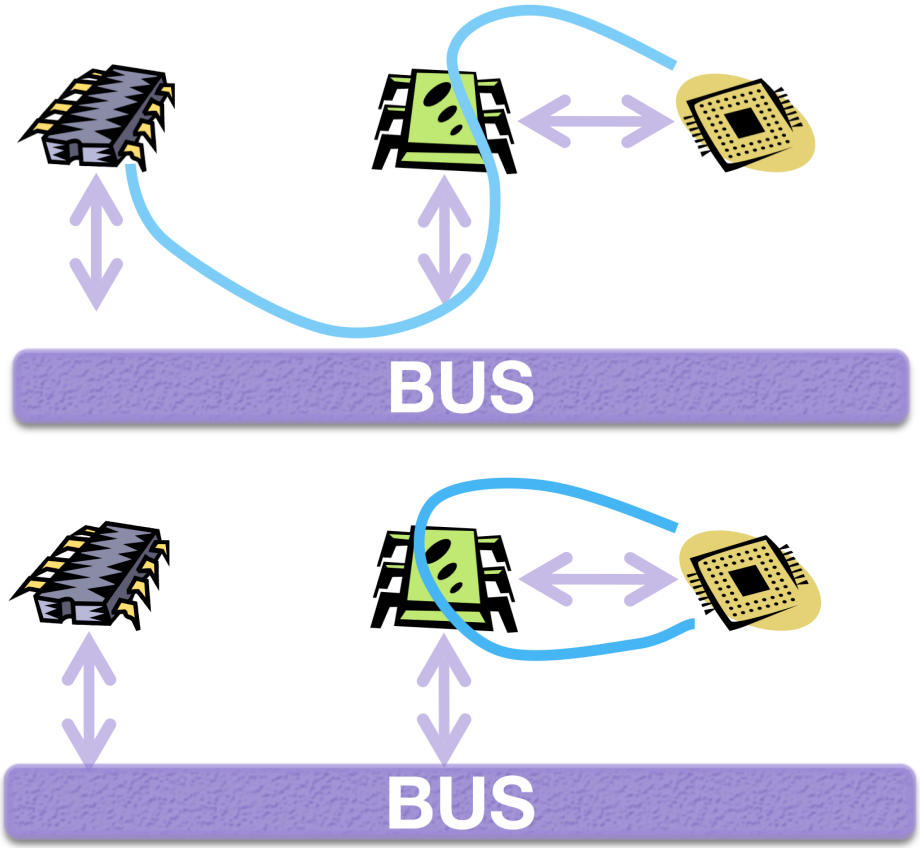
Hardware Caching

- Modern systems benefit from multiple caches (data, instruction, Translation lookaside buffer (TLB), etc.)
- Many caches are hierarchical
- Some of them are physically close to the processor
- Hyper-threads usually have separate L1 cache
- Cores usually have separate L2 cache
- L3 cache is shared



Hardware Caching

- When addressing the memory, the cache checked first for the presence of the cached value
- In case of miss:
  - Access cache
  - Access memory
  - Free cache memory
  - Load data to cache
  - Load data to processor
- In case of hit:
  - Access cache
  - Load data to processor



Cache Performance Metrics

- Miss Rate
  - Fraction of memory references not found in cache
    - 1 - hit rate
  - Typical numbers (in percentages):
    - 3-10% for L1
    - Can be quite small (e.g., 1%) for L2, depending on size, etc.
- Hit Time
  - Time to deliver a line in the cache to the processor
    - Includes time to determine whether the line is in the cache
  - Typical numbers:
    - 1-2 clock cycle for L1
    - 5-20 clock cycles for L2
- Miss Penalty
  - Additional time required because of a miss
  - Typically 50-200 cycles for main memory (Trend: increasing!)

Performance of Computer System

Many different factors to take into account when determining performance:

- Technology
  - Circuit speed (clock, MHz)
  - Processor technology
- Organization
  - Type of processor (ILP)
  - Configuration of the memory hierarchy
  - Type of I/O devices
  - Number of processors in the system
- Software
  - Quality of the compilers
  - Organization and quality of OS, databases, etc

Content of the Class

- Review of the evolution of the speed of computer systems
- The role of performances
- Measuring performances
- How to combine different performance measures to make decisions
- Programs to determine comprehensive performance indexes

Performance

- Performance is determined by execution time
- Do any of the other variables equal performance?
  - # of cycles to execute program?
  - # of instructions in program? (MIPS)
  - # of cycles per second?
  - average # of cycles per instruction?
  - average # of instructions per second?

Common pitfall: Thinking one of the variables is indicative of performance when it really is not.

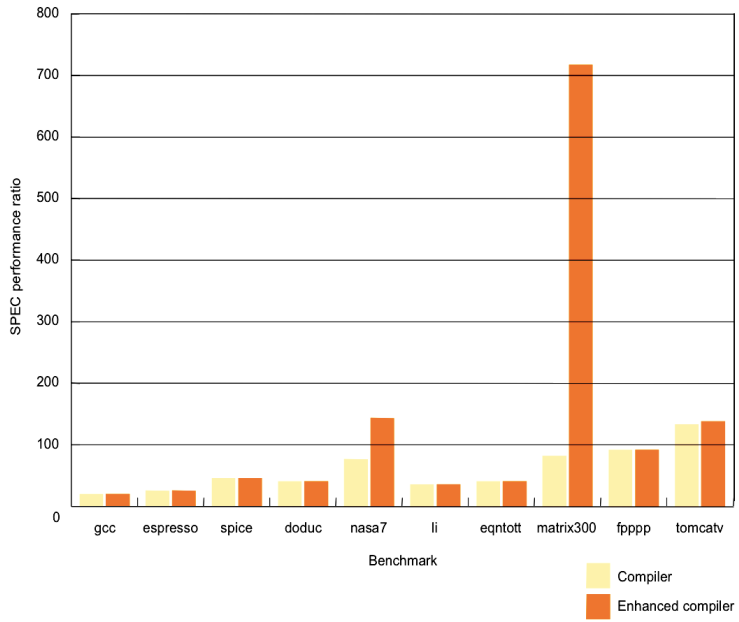
Benchmarks

- Performance best determined by running a real application
  - Use programs typical of expected workload
  - Or, typical of expected class of applications. e.g., compilers/editors, scientific applications, graphics, etc.

- Small benchmarks
  - Nice for architects and designers
  - Easy to standardize
  - It can be abused

Abuse in the SPEC'89

- Matrix 300 (SPEC '89) on an IBM Powerstation 550



  - 99% of execution time is in a single line of code! Designed to test memory system. Compiler performed a one-shot optimization to eliminate cache misses.
  - *Standard Performance Evaluation Corporation (SPEC) `https://www.spec.org/benchmarks.html`

Types of Benchmark

- Real program
- Component Benchmark / Microbenchmark
- Kernel
- Synthetic Benchmark
- I/O Benchmarks
- Database Benchmarks
- Parallel Benchmarks

Popular Benchmark Suites

- Desktop
  - SPEC CPU2017 - CPU intensive, integer floating-point applications
  - SPECapc (2015) - Graphics benchmarks
  - SYSMark (2018), Winstone, Winbench
- Embedded
  - EEMBC Autobenc 2.0 - Collection of kernels from different application areas
  - Dhrystone - Old synthetic benchmark
- Servers
  - SPECweb, SPECfs
  - TPC-C - Transaction processing system
  - TPC-H, TPC-R - Decision support system
  - TPC-W - Transactional web benchmark

Difference of Benchmarks

- CPI is not always the best way to compare the performance, especially when processors use different instruction sets.
- In such cases synthetic benchmarks are preferred (Dhrystone, CoreMark).
- It should be noted that different benchmarks consider different levels of performance.
- Thus, some benchmarks measure only CPU performance without cache performance, and some do system evaluation.
- All these details need to be taken into account for fair comparison. Also, the reported performance of some benchmarks can be sensitive to processor frequency (e.g. performance per MHz).

Amdahl's Law

- Amdhal's law gives the theoretical speedup in latency of the execution of a task at fixed workload that can be expected of a system whose resources are improved.
- This law often used in parallel computing to predict the theoretical speedup when using multiple processors.

Execution time after improvement =
Execution time unaffected +
(Execution time affected / Amount of improvement)

Summary

- Performance is specific to a particular program(s)
  - Total execution time is a consistent summary of performance
- For a given architecture performance increases come from:
  - Increases in clock rate (without adverse CPI affects)
  - Improvements in processor organization that lower CPI
  - Compiler enhancements that lower CPI and/or instruction count
  - ...
- Consider that improvements in one aspect of a machine's performance may affect the total performance in unexpected ways

- You should not always believe everything you read! Read carefully!

Acknowledgements

- This lecture was created and maintained by Muhammad Fahim, Giancarlo Succi and Alexander Tormasov