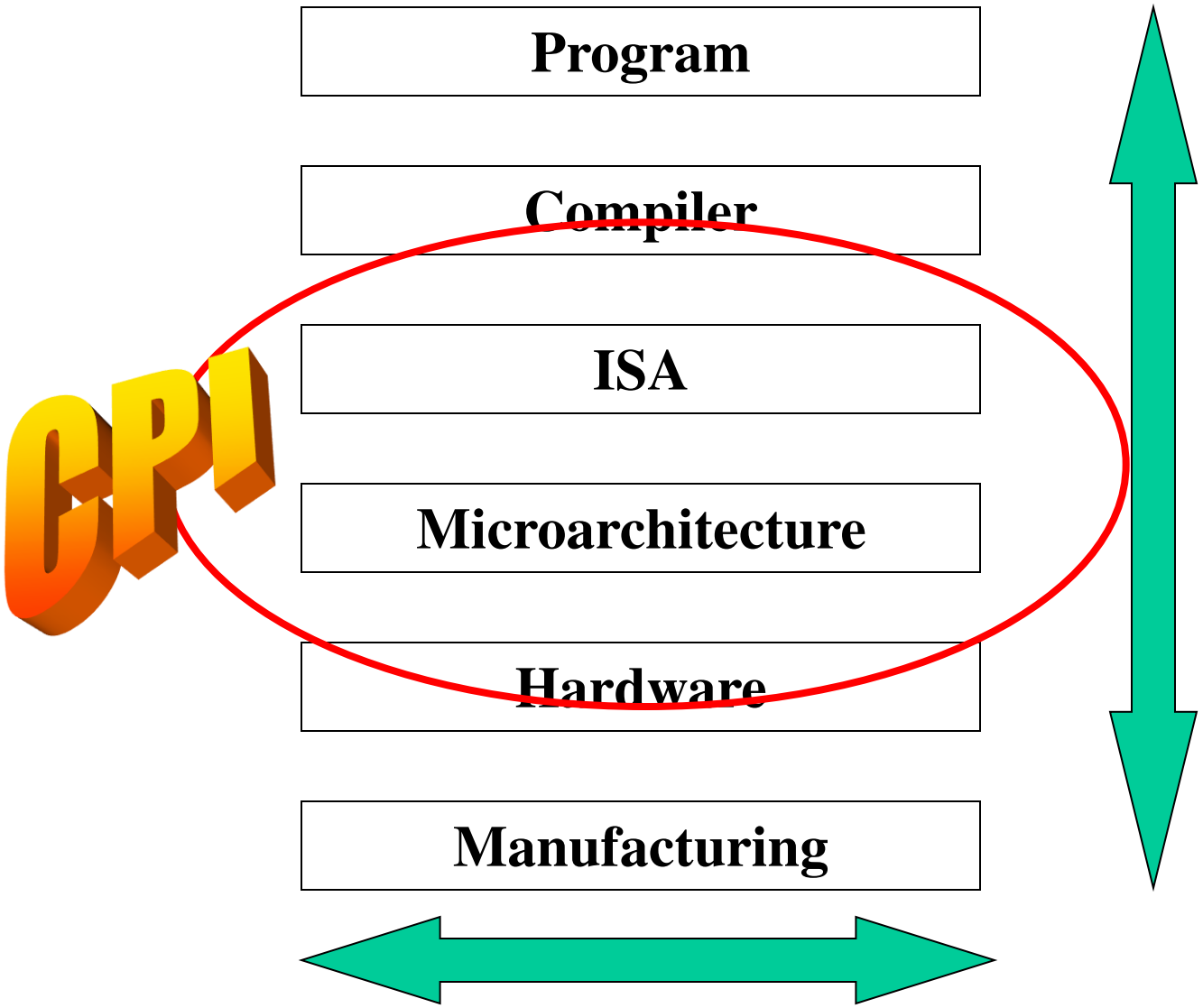
Introduction to Computer Organization

Computer Performance

Overview

- Performance evaluation
- Limitations
- Metrics
- Processor performance equation
- Performance evaluation reports
- Amdahl's law

Performance Evaluation



Performance Evaluation Concepts

- Computer performance evaluation is based on:
  - Throughput (bits per second)
  - Response time (a.k.a. execution time, *elapsed time*)
- Component- and system-level performance
- Processor performance evaluation
  - Based on execution time of a **program:**

$$\textbf{Performance}_X = 1/\textbf{ Execution time}_X$$

Relative performance: $n = \text{Performance}_X / \text{Performance}_Y$

*$n > 1$ => X is n times **faster than** Y*

- Terminology
  - **Improve** performance = **increase** performance
  - **Improve** execution time = **decrease** execution time

Example

- Impact on throughput and response time of:
  - Using a faster processor
    - *Decrease* in *response time*
    - *Increase* in *throughput*
  - Adding more processors to a system
    - Increase in throughput
    - Decrease in response time (*if overhead is low*)
      - In Massively Parallel Processors (MPP)
      - In Symmetric Multiprocessing Processors (SMP)
        - » Only if additional processors reduce queue time

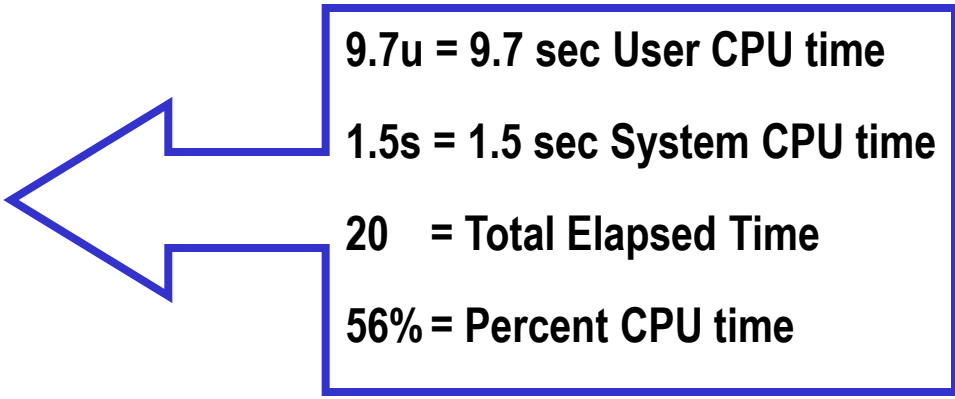
Measuring Performance

Components of the execution time of a program:

1. CPU execution time
   - User CPU time
   - System CPU time
2. I/O time
3. Time spent running other programs

Unix **time** command

```
time cc prog.c
9.7u  1.5s  20  56%
```



CPU Performance Equation

- **CPU time** = CPU clock cycles * cycle time
  = CPU clock cycles / clock rate
  - CPU clock cycles = IC * CPI
    - IC: instruction count (number of instructions per program)
    - CPI: average cycles per instruction

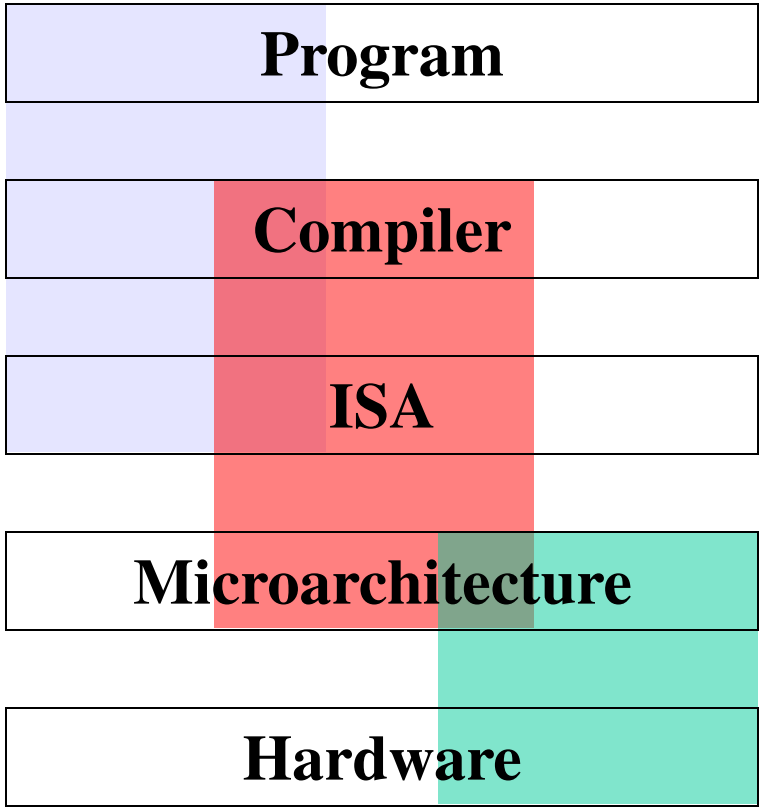
CPU time = IC * CPI * cycle time

$$\frac{\text{seconds}}{\text{program}} = \frac{\text{instructions}}{\text{program}} * \frac{\text{clock cycles}}{\text{instruction}} * \frac{\text{seconds}}{\text{clock cycle}}$$

- **CPU clock cycles** = Σ_i (CPI_i * IC_i)
  - IC_i : count of instructions of class *i*
  - CPI_i : cycles that takes to execute instructions of class *i*

Scope of Performance Sources

CPU time = IC * CPI * Cycle time



Abstraction level interdependence

Example 1

- A program runs in 10 secs on a 2.0 GHz processor. A designer wants to build a new computer that can run the program in 6 secs by increasing the clock frequency. However the average "new" CPI will be 1.2 times higher.
- What faster clock rate should the designer use?

$$\frac{10}{6} = \frac{\text{IC} * \text{CPI} / 2\ \text{GHz}}{\text{IC} * 1.2\ \text{CPI} / X\ \text{GHz}}$$

(current exec. time)

(target execution time)

Solve for X, to obtain **X = ___ GHz**

Example 2

- **Comparing two compiler code segments**

Instruction class *i*	CPI_i for instruction class *i*
A	1
B	2
C	3

Code sequence	Instruction counts (IC_i) 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? $S_1 = 2 \cdot 1 + 1 \cdot 2 + 2 \cdot 3 = 10$ cyc

$S_2 = 4 \cdot 1 + 1 \cdot 2 + 1 \cdot 3 = 9$ cyc

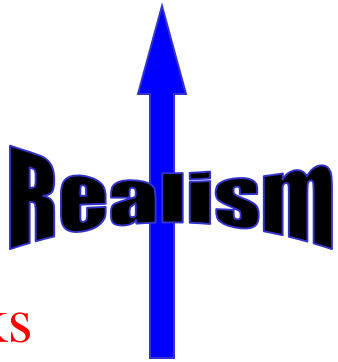
Components of the CPU Equation

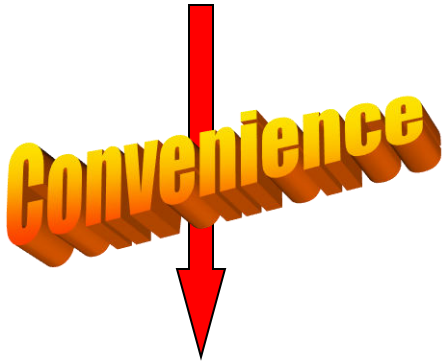
- **IC** Instruction count
  - Compiler
    - Instruction set simulator
    - Execution-based monitoring (*profiling*)
- **CPI** if pipelined execution is used

  CPI_i = Pipeline CPI_i + Memory CPI_i
- **Clock cycle time**
  - Timing estimators or verifiers (complete design)
  - Target cycle time

Performance Evaluation Programs

- Ideal situation: known programs (*workload*)
- Benchmarks
  - **Real programs**
  - Kernels
  - Toy benchmarks
  - Synthetic benchmarks





- Risk: adjust design to benchmark requirements
  - (partial) solution: use **real programs**
    - Engineering or scientific applications
    - Software development tools
    - Transaction processing
    - Office applications

Performance Reports

- **Reproducibility**
  - Include hardware / software configuration
  - Evaluation process conditions
- **Summarizing performance**
  - *Total time*: CPU time + I/O time + Other time
  - *Arithmetic mean*: AM = $1/n * \Sigma$ exec $time_i$
  - *Harmonic mean*: HM = $n / \Sigma\ (1/rate_i)$
  - *Weighted mean*: WM = $\Sigma\ w_i$ * exec $time_i$
  - *Geometric mean*: GM = $(\Pi$ exec time $ratio_i)^{1/n}$

$$\frac{GM(X_i)}{GM(Y_i)} = \left[\frac{X_i}{Y_i}\right]$$

Arithmetic and Geometric Means

	A	B	C
P1	1	10	20
P2	1000	100	20

Execution time (in seconds)
machines: A, B, and C
programs: P1 and P2

	Normalized to A			Normalized to B			Normalized to C		
	A	B	C	A	B	C	A	B	C
Program P1	1.0	10.0	20.0	0.1	1.0	2.0	0.05	0.5	1.0
Program P2	1.0	0.1	0.02	10.0	1.0	0.2	50.0	5.0	1.0
Arithmetic mean	1.0	5.05	10.01	5.05	1.0	1.1	25.03	2.75	1.0
Geometric mean	1.0	1.0	0.63	1.0	1.0	0.63	1.58	1.58	1.0
Total time	1.0	0.11	0.04	9.1	1.0	0.36	25.03	2.75	1.0

Amdahl's Law

- Law of diminishing returns

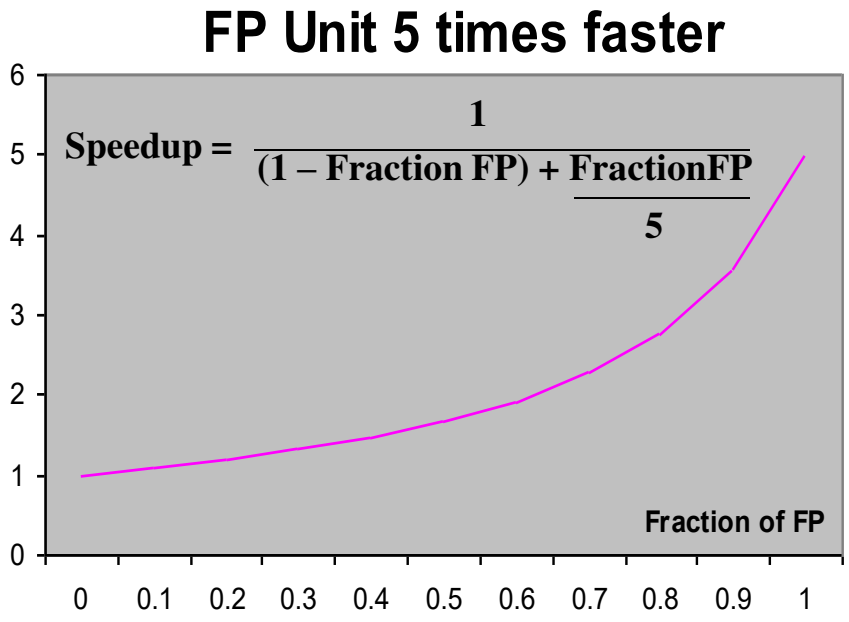
$$\text{Speedup} = \frac{\text{Execution time before improvement}}{\text{Execution time after improvement}}$$

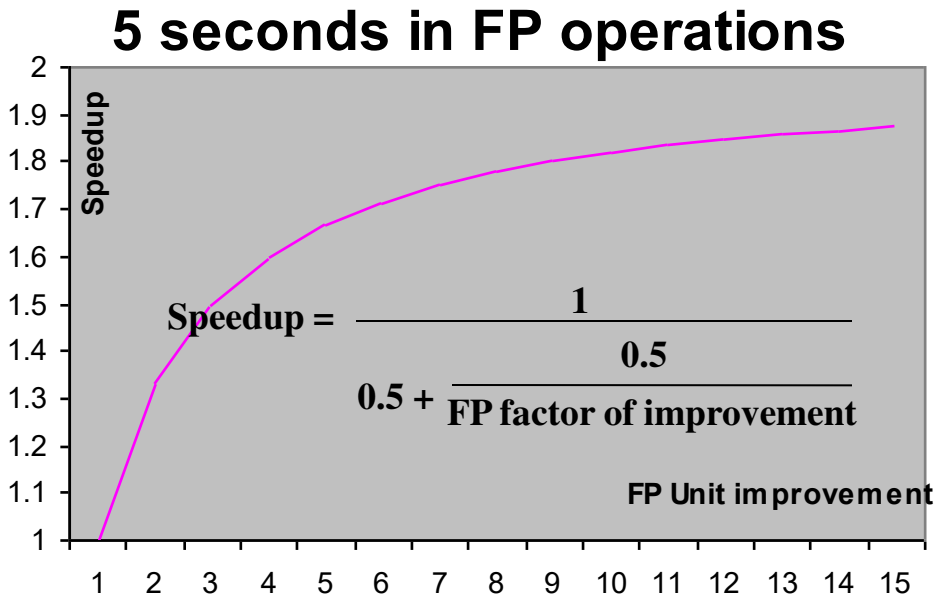
Example: Two factors - **F1**: 75% improve, **F2**: 50% improve

$$\text{Speedup} = \frac{1}{(1 - \text{fraction enhanced}) + (\text{fraction enhanced}/\text{factor of improvement})}$$

Example

- A program runs in 10 seconds
- What is the speedup after a faster floating point unit is incorporated?





Conclusions

- **Many different performance data**
  - CPU time, I/O time, Other time, Total time
- **Select best presentation method**
  - Arithmetic mean for execution times
  - Geometric mean for performance ratios

- **Watch out for Amdahl's Law**
  - Diminishing returns w/ improved performance
  - Impacts 

of development