Lecture 5

Computer Architecture
(Spring 2020)

Quantitative Approach

Dr. Duo Liu (刘铎)
Office: Main Building 0626
Email: liuduo@cqu.edu.cn

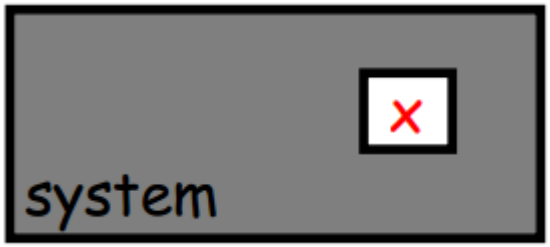
“Make the Common Case Fast”

- “the most important, pervasive, and simple principle of computer design”
  - in making a design trade-off…
    - favor the frequent case rather than infrequent case
  - when determining how to allocate resources…
    - favor the frequent event rather than the rare event
  - when optimizing the design of a module…
    - target the average functional behavior
- …besides, the frequent case is often simpler

1. How to determine what the frequent case is?
2. How to determine the amount of the possible performance gain in making the frequent case faster ?

Amdahl's Law

- What is the overall speedup after improving a component x of a system?



$$speedup = \frac{originalExecutionTime}{newExectionTime} = \frac{newPerformance}{originalPerformance}$$

- If component x is improved by Sx and component x affects a fraction Fx of the overall execution time then

$$speedup = \frac{1}{(1 - Fx) + Fx / Sx)}$$

Fx /Sx： new exec. time of improved part

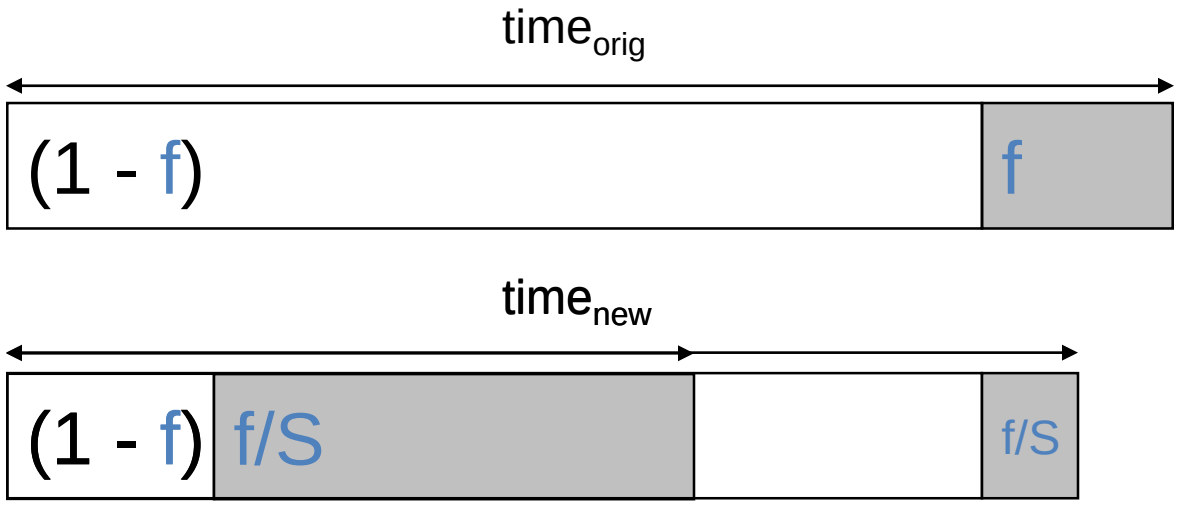
Amdahl's Law

Speedup = $\text{time}_{\text{without enhancement}} / \text{time}_{\text{with enhancement}}$

An enhancement speeds up fraction f of a task by factor S

$$\text{time}_{\text{new}} = \text{time}_{\text{orig}} \cdot ((1-f) + f/S)$$

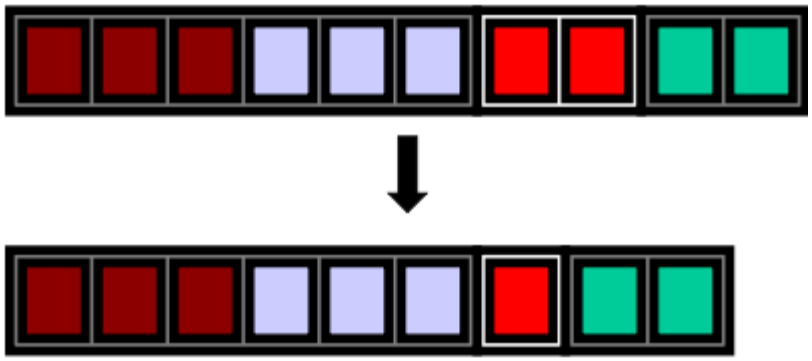
$$S_{\text{overall}} = 1 / ((1-f) + f/S)$$



Amdahl's Law - Example

$$speedup = \frac{1}{(1 - Fx) + Fx / Sx)}$$

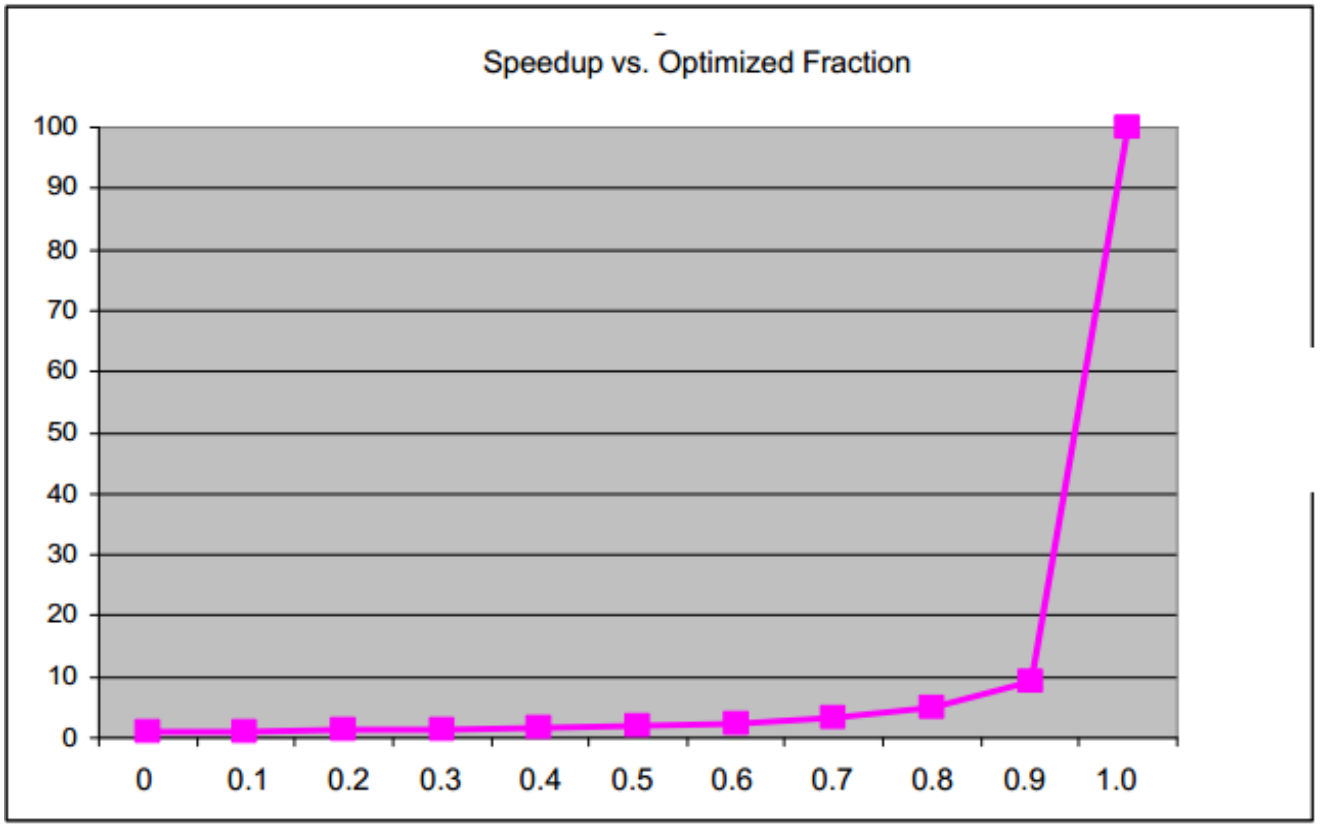
If we optimize the module for the floating-point instructions by a factor of 2, but the system will normally run programs with only 20% of floating point instructions then the speedup is only



$$speedup = \frac{1}{(1 - 0.2) + 0.2 / 2)} = \frac{1}{0.9} = 1.111$$

Amdahl's Law - Example

- If Sx=100, what is the overall speedup as a function of Fx?



Amdahl's Law and the Law of Diminishing Returns

- the closer to 1 is Fx, the closer to Sx is the overall speedup…
  - i.e. [make common case fast]
- however, as Sx → ∞, speedup → 1 / (1- Fx)
  - i.e., once Fx/Sx is small with respect to (1-Fx) the price/performance ratio falls rapidly as Sx is increased
- the incremental improvement in speedup gained by an additional improvement in the performance of just a portion of the computation diminishes as improvements are added

Amdahl's Law - Example

- **Example: H&P 5th P47**

Suppose that we want to enhance the processor used for Web serving. The new processor is 10 times faster on computation in the Web serving application than the original processor. Assuming that the original processor is busy with computation 40% of the time and is waiting for I/O 60% of the time, what is the overall speedup gained by incorporating the enhancement?

- **Answer**

$$\text{Fraction}_{\text{enhanced}} = 0.4;\ \text{Speedup}_{\text{enhanced}} = 10;\ \text{Speedup}_{\text{overall}} = \frac{1}{0.6 + \frac{0.4}{10}} = \frac{1}{0.64} \approx 1.56$$

CPU Time

- CPU Time
  - user CPU Time
    - spent in the user program
  - system CPU Time
    - spent in the OS performing tasks required by the program
    - harder to measure and to compare across architectures
  - CPU performance = user CPU time on an unloaded system

CPU Time = (Clock Cycles for a Program) x (Clock Cycle Time)

= (Clock Cycles for a Program) / (Clock Frequency)

- most computers run with a single clock signal (strictly synchronous design) whose discrete time events are called cycles, periods, or ticks
  - a μP with a 1ns clock period runs at 1GHz of clock frequency…

CPU Time – Three Main Factors

CPU Time = (Clock Cycles for a Program) x CCT

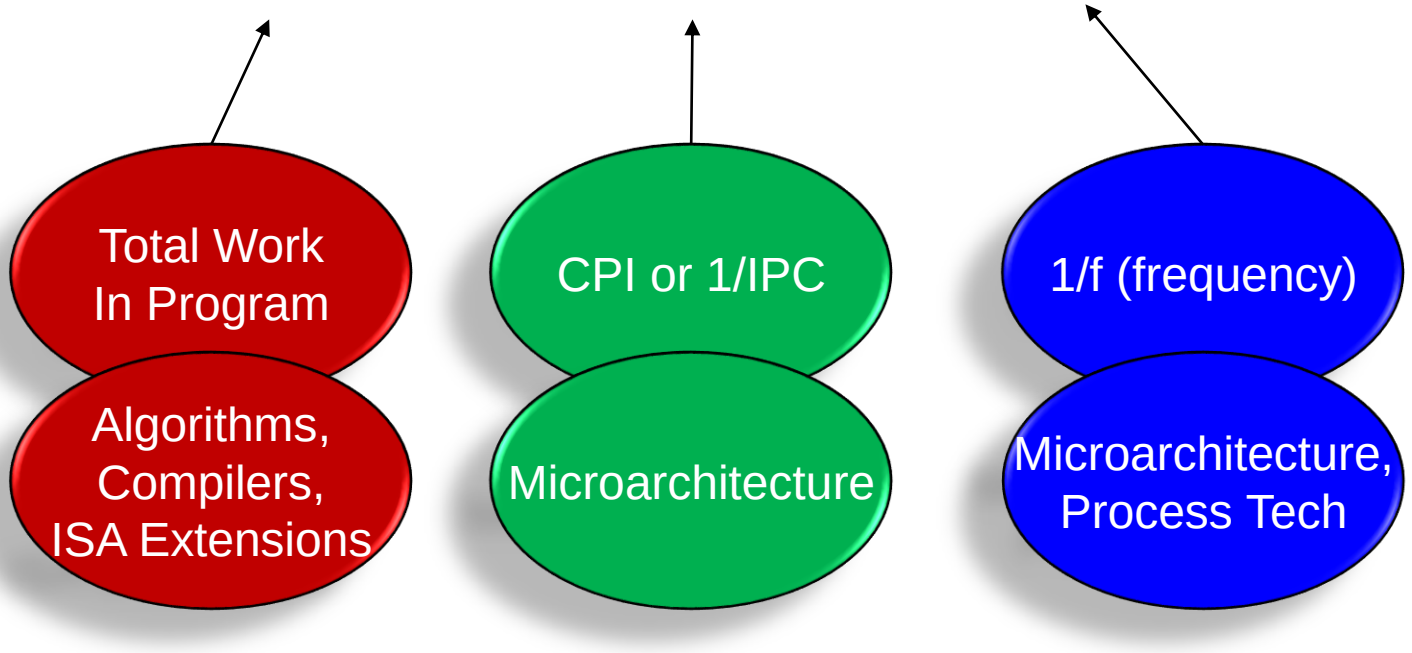
- IC = instruction count
  - number of instructions executed for a program
- CPI = clock cycles per instruction
  - average number of clock cycles per instruction of a program
- its reciprocal is **IPC = instruction per clock cycles**

CPU Time = IC x CPI x CCT

- CPU Time equally depends on these three factors
  - a 10% improvement in any of these leads to a 10% improvement in CPU time

The *Iron Law* of Processor Performance

$$\frac{Time}{Program} = \frac{Instructions}{Program} \times \frac{Cycles}{Instruction} \times \frac{Time}{Cycle}$$



Architects target CPI, but ***must*** understand the others

CPU Time - Dependencies

CPU Time = IC x CPI x CCT

	IC	CPI	CCT
Program	√		
Compiler	√		
ISA	√	√	
HW organization		√	√
HW technology			√

- some interdependencies, but many techniques improve a single factor

Improving Performance by Exploiting Parallelism

- at the system level
  - use multiple processors, multiple disks
    - scalability is key to adaptively distribute workload in server apps
- at the single microprocessor level
  - exploit instruction level parallelism (ILP)
    - e.g., pipelining overlaps the execution of instruction to reduce the overall program CPU Time
      - reduces CPI by overlapping instructions in time
      - possible because many subsequent instructions are independent
    - e.g. parallel computation
      - reduces CPI by overlapping instructions in space
      - duplicate hardware modules such as ALUs
- at the circuit level
  - carry-lookahead adders speed-up sums
    - from linear to logarithmic

CPU Time – broken down per instruction

CPU Time = IC x CPI x CCT

$$\text{CPU Time} = \sum_i \left(IC_i \times CPI_i \right) \times CCT$$

$$CPI = \sum_i \left(\frac{IC_i}{IC} \times CPI_i \right) = \sum_i \left(IF_i \times CPI_i \right)$$

- frequent instructions have larger contributions on CPI
- CPI should be measured to include pipeline/memory effects
  - it is not sufficient to calculate it from the reference manual table
- NOTE: it is ok to compare two designs based only on CPI (or IPC) only if IC and CCT are the same!

CPU Time Example

- **Example: H&P 5th P50**

Suppose we have made the following measurements:

Frequency of FP operations = 25%

Average CPI of FP operations = 4.0

Average CPI of other instructions = 1.33

Frequency of FPSQR = 2%

CPI of FPSQR = 20

Assume that the two design alternatives are to decrease the CPI of FPSQR to 2 or to decrease the average CPI of all FP operations to 2.5. Compare these two design alternatives using the processor performance equation.

CPI Example

- Alternative compiled code sequences using instructions in classes A, B, C. What is avg. CPI?

Class	A	B	C
CPI for class	1	2	3
IC in sequence 1	2	1	2
IC in sequence 2	4	1	1

- Sequence 1: IC = 5
  - Clock Cycles
    = 2×1 + 1×2 + 2×3
    = 10
  - Avg. CPI = 10/5 = 2.0
- Sequence 2: IC = 6
  - Clock Cycles
    = 4×1 + 1×2 + 1×3
    = 9
  - Avg. CPI = 9/6 = 1.5

Example: Average Instruction Execution Time

- Assuming a simple un-pipelined processor with CCT = 2ns

Operation	IFi	CPIi	IFi x CPIi
ALU	0.5	4	2
Load	0.2	5	1
Store	0.1	5	0.5
Branch	0.2	4	0.8

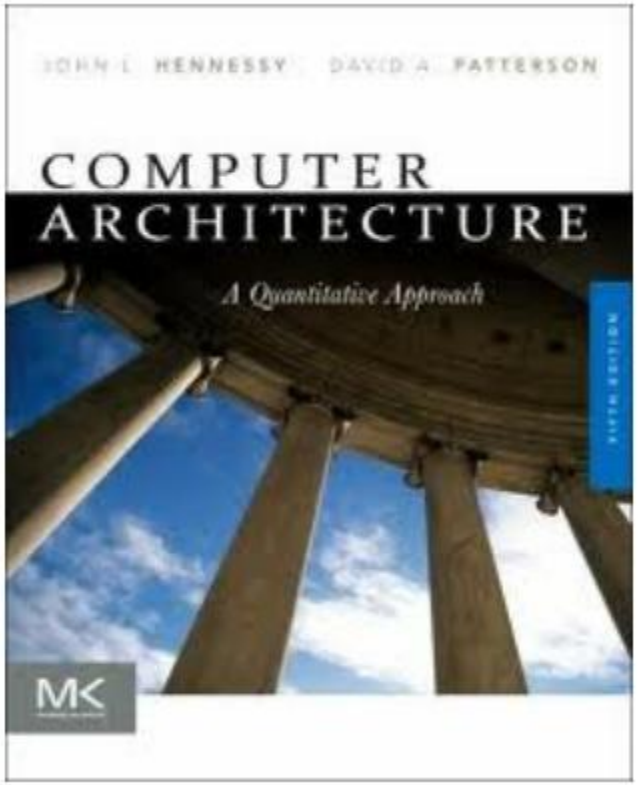
- $CPI = \sum_i (IFi \times CPIi) = 4.3$
- Average instruction execution time = CPI x CCT = 8.6ns

Assigned Readings

- Computer Architecture – A Quantitative Approach by
  John Hennessy
  - Stanford University

  Dave Patterson
  - UC Berkeley

  Fifth Edition - 2012
  Morgan Kaufmann (Elsevier)



- Read Sections 1.8-1.12

Homework #1

H&P 5th

Page 67: 1.15 and 1.16

Due 2020/03/26