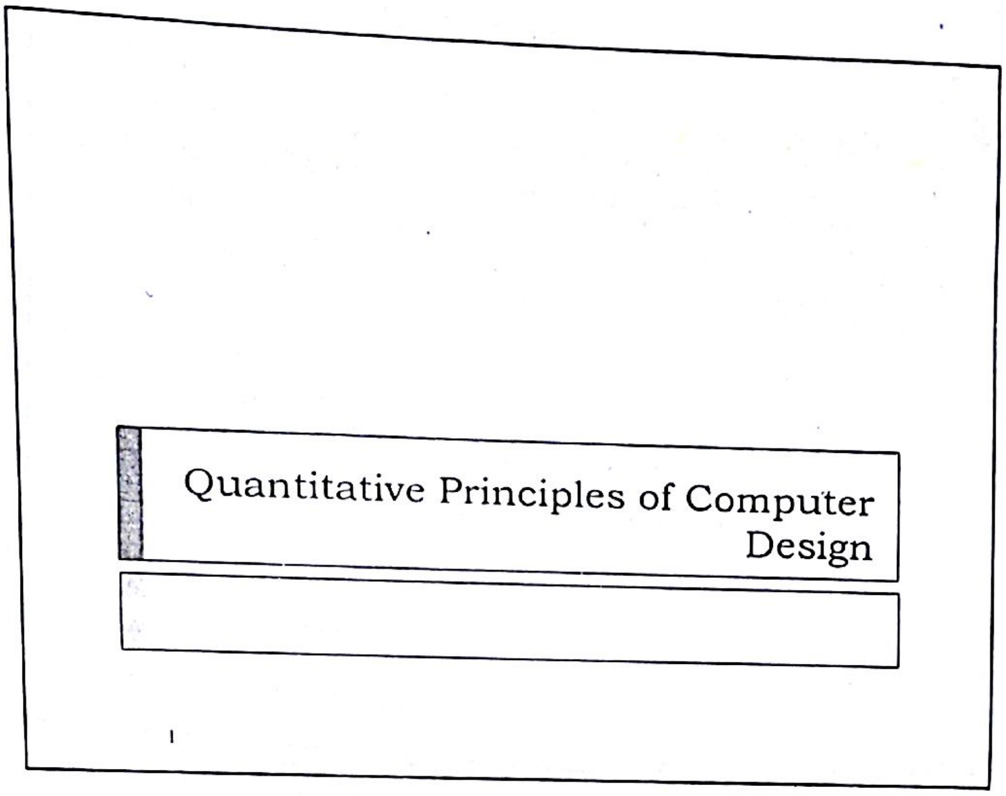
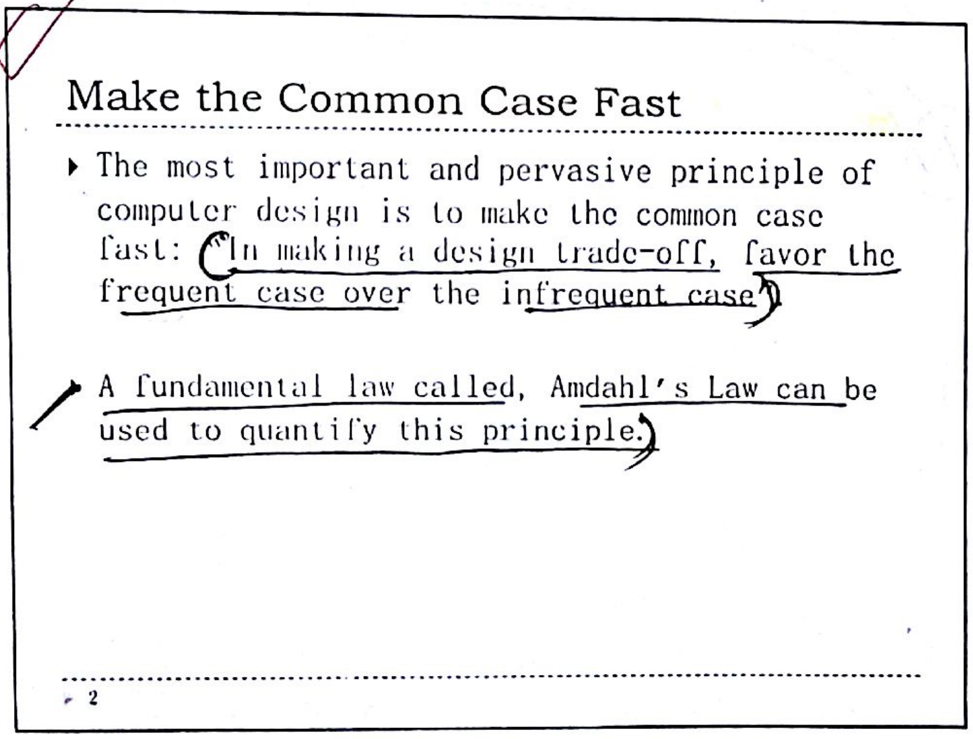
Computer Architecture.

[illegible]
MSA
2/27/2016



Quantitative Principles of Computer Design

1



Make the Common Case Fast

- The most important and pervasive principle of computer design is to make the common case fast: "In making a design trade-off, favor the frequent case over the infrequent case".
- A fundamental law called, Amdahl's Law can be used to quantify this principle.

2

Amdahl's Law

- The performance improvement can be obtained by improving some portion of a computer can be calculated using Amdahl's Law.

- Amdahl's Law states that – "*The performance improvement to be gained from using some faster mode of execution is limited by the fraction of the time the faster mode can be used.*"

Speedup

- Amdahl's Law defines the *speedup* that can be gained by using a particular feature. Speedup is the ratio,

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

Alternatively,

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

Speedup

- Speedup depends on two factors:
  1. The fraction of computation time in the original computer that can be converted to take advantage of the enhancement.

     $\text{Fraction}_{\text{enhanced}} \le 1$

  2. The improvement gained by the enhancement execution mode; that is how much faster the task would run if the enhancement mode were used for the entire program.
     1. $\text{Speedup}_{\text{enhanced}} > 1$

Some Equation for math

$$\text{Execution time}_{\text{new}} = \text{Execution time}_{\text{old}} \times \left((1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}} \right)$$

The overall speedup is the ratio of the execution times:

$$\text{Speedup}_{\text{overall}} = \frac{\text{Execution time}_{\text{old}}}{\text{Execution time}_{\text{new}}} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$

Example

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?

- Ans:

$\text{Fraction}_{\text{enhanced}} = 40\% = 0.4$

$\text{Speedup}_{\text{enhanced}} = 10$

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

7

Example 2

Example A common transformation required in graphics processors is square root. Implementations of floating-point (FP) square root vary significantly in performance, especially among processors designed for graphics. Suppose FP square root (FPSQR) is responsible for 20% of the execution time of a critical graphics benchmark. One proposal is to enhance the FPSQR hardware and speed up this operation by a factor of 10. The other alternative is just to try to make all FP instructions in the graphics processor run faster by a factor of 1.6; FP instructions are responsible for half of the execution time for the application. The design team believes that they can make all FP instructions run 1.6 times faster with the same effort as required for the fast square root. Compare these two design alternatives.

Answer We can compare these two alternatives by comparing the speedups:

$$\text{Speedup}_{\text{FPSQR}} = \frac{1}{(1-0.2) + \frac{0.2}{10}} = \frac{1}{0.82} = 1.22$$

$$\text{Speedup}_{\text{FP}} = \frac{1}{(1-0.5) + \frac{0.5}{1.6}} = \frac{1}{0.8125} = 1.23$$

8

Usefulness of Amdahl's Law

- It helps to get a rough estimate of performance improvement. nearly calculation
- It gives a simple relationship between the nature of performance improvement and the problem characteristics.
- We can make an enhancement to a machine that will improve performance when it is used.

The CPU Performance Equation

Derive the CPU performance equation:

CPU time is the amount of time for which a CPU is used for processing instructions of a computer program or OS. CPU time of a program can be expressed as:

CPU time = CPU clock cycles for a program × Clock cycle time (1)

Or

$$\text{CPU time} = \frac{\textit{CPU clock cycle for a program}}{\textit{Clock rate}} \quad (2)$$

If we know the number of clock cycles and the instruction count (IC), we can calculate the average number of clock cycle per instruction (CPI).

$$\text{CPI} = \frac{\textit{CPU clock cycle for a program}}{\textit{Instruction count}} \quad (3)$$

10

Thus clock cycle can be defined as IC × CPI. This replaces the formula (1) as,

CPU time = Instruction count × Clock cycle time × CPI (4)

$$\text{Or CPU time} = \frac{\textit{Instruction count} \times \textit{CPI}}{\textit{Clock rate}} \quad (5)$$

CPU clock cycles can be calculated as,

$$\text{CPU clock cycles} = \sum_{i=1}^{n} \text{IC}_i \times \text{CPI}_i$$

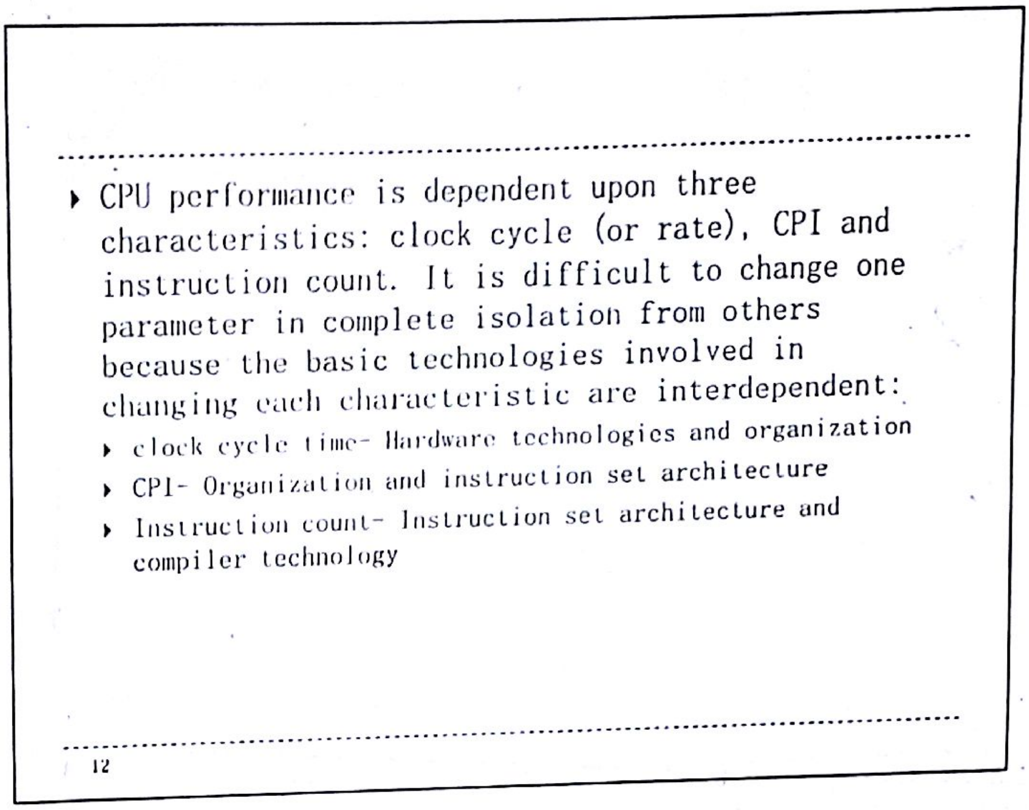
Where IC_i represents number of times instruction i is executed in a program and CPI_i represents the average number of instructions per clock for instruction i. This form can be used to express CPU time as,

$$\text{CPU time} = \left(\sum_{i=1}^{n} \text{IC}_i \times \text{CPI}_i\right) \times \text{Clock cycle time}$$

And overall CPI as

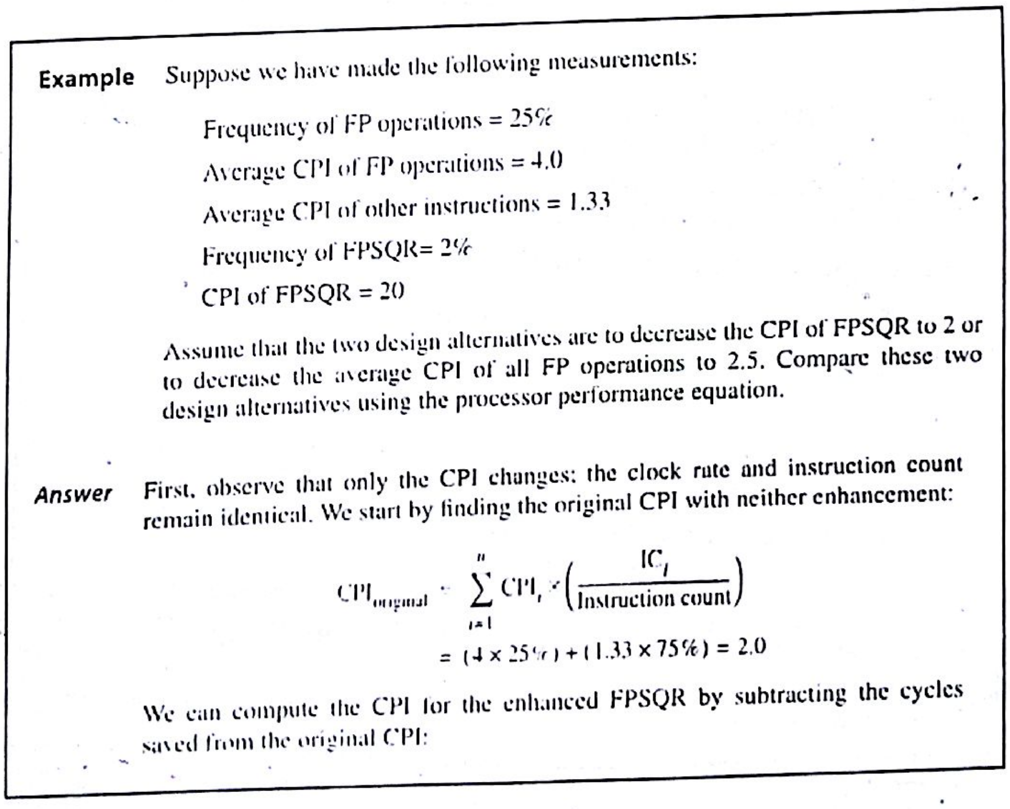
$$\text{CPI} = \frac{\sum_{i=1}^{n} \text{IC}_i \times \text{CPI}_i}{\text{Instruction count}} = \sum_{i=1}^{n} \frac{\text{IC}_i}{\text{Instruction count}} \times \text{CPI}_i$$

11



- CPU performance is dependent upon three characteristics: clock cycle (or rate), CPI and instruction count. It is difficult to change one parameter in complete isolation from others because the basic technologies involved in changing each characteristic are interdependent:
  - clock cycle time- Hardware technologies and organization
  - CPI- Organization and instruction set architecture
  - Instruction count- Instruction set architecture and compiler technology

12



Example 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%

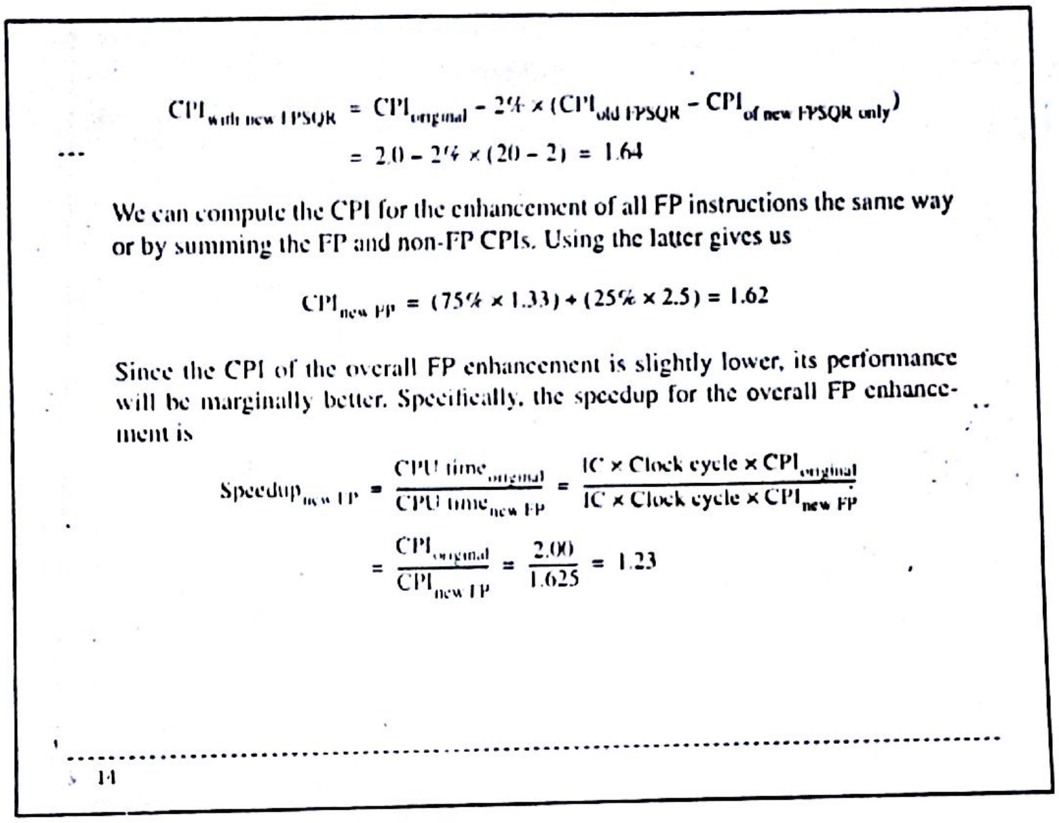
CPI of FPSQR = 20

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.

Answer First, observe that only the CPI changes; the clock rate and instruction count remain identical. We start by finding the original CPI with neither enhancement:

$$\text{CPI}_{\text{original}} = \sum_{i=1}^{n} \text{CPI}_i \times \left(\frac{\text{IC}_i}{\text{Instruction count}}\right)$$
$$= (4 \times 25\%) + (1.33 \times 75\%) = 2.0$$

We can compute the CPI for the enhanced FPSQR by subtracting the cycles saved from the original CPI:



$$CPI_{\text{with new FPSQR}} = CPI_{\text{original}} - 2\% \times (CPI_{\text{old FPSQR}} - CPI_{\text{of new FPSQR only}})$$
$$= 2.0 - 2\% \times (20 - 2) = 1.64$$

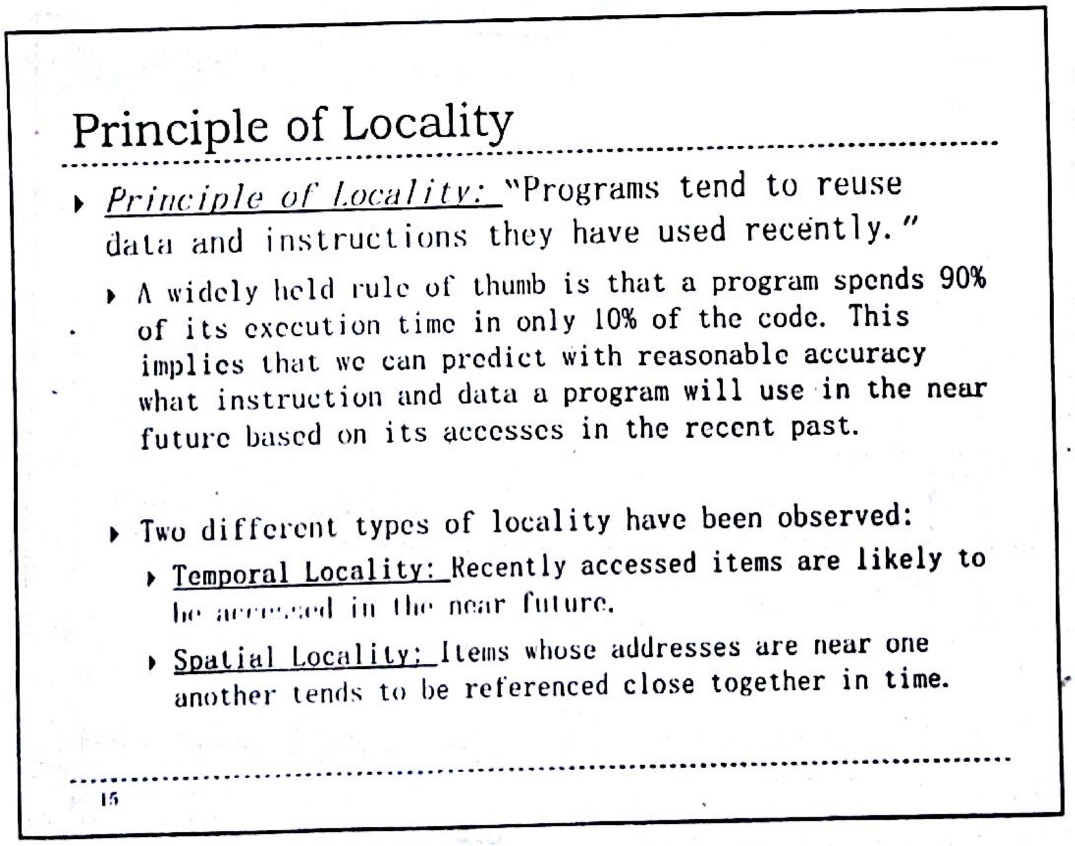
We can compute the CPI for the enhancement of all FP instructions the same way or by summing the FP and non-FP CPIs. Using the latter gives us

$$CPI_{\text{new FP}} = (75\% \times 1.33) + (25\% \times 2.5) = 1.62$$

Since the CPI of the overall FP enhancement is slightly lower, its performance will be marginally better. Specifically, the speedup for the overall FP enhancement is

$$\text{Speedup}_{\text{new FP}} = \frac{\text{CPU time}_{\text{original}}}{\text{CPU time}_{\text{new FP}}} = \frac{\text{IC} \times \text{Clock cycle} \times CPI_{\text{original}}}{\text{IC} \times \text{Clock cycle} \times CPI_{\text{new FP}}}$$
$$= \frac{CPI_{\text{original}}}{CPI_{\text{new FP}}} = \frac{2.00}{1.625} = 1.23$$

14



Principle of Locality

- *Principle of Locality:* "Programs tend to reuse data and instructions they have used recently."
  - A widely held rule of thumb is that a program spends 90% of its execution time in only 10% of the code. This implies that we can predict with reasonable accuracy what instruction and data a program will use in the near future based on its accesses in the recent past.
  - Two different types of locality have been observed:
    - Temporal Locality: Recently accessed items are likely to be accessed in the near future.
    - Spatial Locality: Items whose addresses are near one another tends to be referenced close together in time.

15