# Measuring Performance



- \*\* What is Performance?
- \*\* Performance Measurement Methods
  - **•**MIPS
  - •CPI / IPC
  - Choosing Programs to Evaluate Performance
  - Benchmark Suits
- \* Speedup
- Representative Principles of Computer Design



- \*\* Performance means
  - How quickly a given system executes a program/ a set of programs
- **\*** Two aspects
  - Response Time
    - The time between the start and the completion of an event—execution time.
  - Throughput
    - The total amount of work done in a given time



- **\*\*** Millions of Instructions Per Second
- \* An early measure of computer performance.

MIPS = <u>No. of instruction executed in a program</u>

Time required to run the program

typically expressed in millions of instructions per second.

- \*\* Out Of Use
  - It does not account for the fact that different systems often require different no.s of instructions to implement a given
  - program.

A computer's MIPS ratings does not tell you about how many instructions it require to perform a given task.

#### CPI / IPC

- ₩ CPI
  - Cycle Per Instruction
  - •No of clock cycles required to execute each instruction

CPI = No. of clock cycles required

No. of instructions executed

**\*\*** IPC

紫

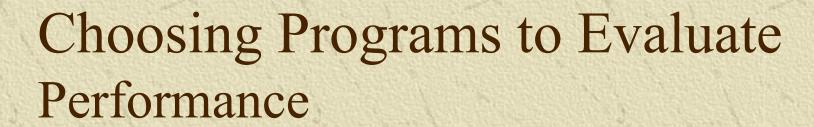
- Instructions Per Cycle
- No. of instructions executed per cycle

IPC = <u>No of Instructions</u>

<u>executed</u> No of clock cycles

required

Large IPC tends to indicate good performance & large CPI indicates poor performance



- # Five levels of programs used to evaluate performance are
  - •Real applications
    - Compilers, Text-processing S/W like Word, and Photoshop.
  - Modified (or scripted) applications
    - To enhance portability or to focus on one particular aspect of system performance
  - Kernels
    - To isolate performance of individual features of a machine.
  - Toy Benchmarks
    - Typically between 10- 100 lines if code and produce a result the user already knows.
  - Synthetic benchmarks
    - Try to match the average frequency of operations and operands of a large set of programs.



- \*A set of programs that are believed to be typical of the programs that will be run on the system
- System is checked for how long it take to execute all of the programs in the suite.
  - For example SPEC benchmark by Standard
- \*\* Performance Evaluation Cooperation
  - Other benchmarks are:
    - Business Winstone
    - CC WindowsWinbench



- Speedup tells how the performance of an architecture changes as different improvements are made to architecture
- Ratio of the execution times before and after a change is made

Speedup = <u>Execution time old(before)</u> <u>Execution time new(after)</u>



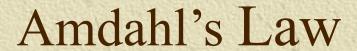
# Quantitative Principles of Computer Design (Continued)

Some principles that are useful in design and analysis of computers

- Make The Common Case Faster
- Amdahl's Law
- **◆**CPU Performance Equation
- Principles Of Locality
- Taking Advantage of Parallelism



- \* In making a design trade-off,
  - Favor the frequent case over the infrequent case because
  - The frequent case is often simpler and can be done faster than the infrequent case
    - e.g. when multiplying two numbers, overflow is the infrequent case, no overflow is the frequent.
    - However, system will slow down when overflow occurs.
  - We have to decide what the frequent case is and how much performance can be improved by making that case faster
  - Amdahl's law quantify this principle



- Impact of a given performance improvement on overall performance depends on
  - How much the improvement improves the performance
  - (when in use)

How often the improvement is in use

Execution Time <sub>new</sub> = Execution Time <sub>old</sub> Fraction <sub>unused</sub> +Fraction <sub>used</sub>

Speedup used

Where

Fraction <sub>unused</sub> = Fraction of time( not instructions) that the improvement is not in use.

Fraction <sub>used</sub> = Fraction of time that the improvement is in use

Speedup <sub>used</sub> = Speedup that occurs when the improvement is used



- \*\* Fraction used and Fraction unused are computed using the execution time before the modification is applied
- \* Speedup can be defined as

$$Speedup = \underbrace{Execution \ time}_{old(before)}$$

$$Execution \ time_{new(after)}$$

$$- \quad \frac{1}{4}$$

$$= Fraction_{unused} + \underbrace{Fraction}_{used}$$

$$Speedup_{used}$$

#### Amdahl's Law

(Continued)

- # The law states
  - "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 = <u>Performance for entire task using the enhancement</u>
  - Performance for entire task without using the enhancement

Or

Speedup = Execution time for the entire work without using the enhancement

Execution time for entire task using the enhancement when possible

 $Speedup = Execution time_{old(before)}$ 

Execution time new(after)

$$=$$
  $\underline{1}$   $(1-Fraction_{enhanced})+Fraction_{enhanced}$ 

Speedup enhanced

#### An Example

\* Problem.

An enhancement is made to a processor of a server system. The new CPU time is 20 times faster than original. If the original CPU is busy with computation 35% of the time and is waiting for I/O 65% of the time, what is the overall speedup gained by incorporating the

enhancement.

Solution.

Fraction 
$$_{enhanced} = 35/100 = .35$$
.  
Speedup  $_{enhanced} = 20$ .  
= 4  
 $(1-0.35)+0.35$   
 $= 1.498173$ 

### **CPU Performance Equation**

```
CPU time = CPU clock cycles for a program x Clock cycle time
(1)
   Clock Cycle Time =
                        Clock rate
(1) Implies
                          cycles for a program ----(2)
 CPU time = CPU clock
                      Clock rate
             CPI = \underline{CPU \ clock \ cycles \ for \ a}
Now,
             program Instruction Count
       CPU clock cycle for a program = CPI \times IC
Now (1) implies
CPU time = CPI x IC x Clock Cycle time -----(3)
```

### CPU Performance Equation

(Continued)

By putting values equation (3) becomes

紫

Seconds

Program

Unfortunately, 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 = H/W technology and organization

CPI = Organization and instruction set

## Principles of Locality

\*\* Programs tend to reuse data and instructions they have used recently

A program spends 90% of its execution time in only 10% of the

\_ code

We can predict with reasonable accuracy what instructions and data a program will use in the near future based on its accesses in the recent past.

Two types of locality

Temporal locality

• States that recently accessed items are likely to be accessed in the near future.

Spatial Locality

• Items whose addresses are near one another tend to be referenced close together in time.

17





- Cone of the most important methods for improving performance It is obtained through
  - ◆ Pipelining to overlap the execution of instructions, so as to reduce the total time to complete a sequence of instructions.



Research is carried out on various aspects in order to improve performance and reduce cost