Measuring and Reporting Performance (Module 3 of Syllabus)

- *The administrator of a large data processing*
  - center may be interested in increasing *throughput—the total amount of work* done in a given time.

- we often want to relate the performance of two different computers, say, X and Y. The phrase "X is faster than Y" is used
  - to mean that the response time or execution time is lower on X than on Y for the given task.
  - "X is *n times faster than Y" will mean*
  - *n= Execution Time$_y$/ Execution Time$_x$*

Why know about performance

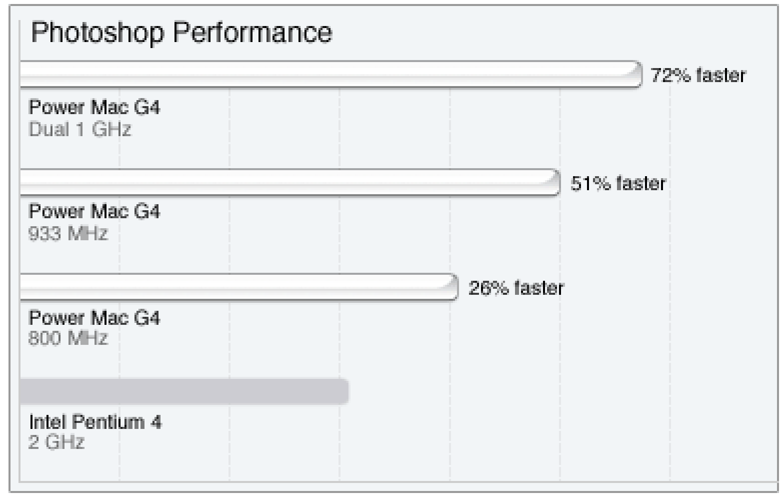
- Purchasing Perspective:
  - Given a collection of machines, which has the
    - Best Performance?
    - Lowest Price?
    - Best Performance/Price?
- Design Perspective:
  - Faced with design options, which has the
    - Best Performance Improvement?
    - Lowest Cost?
    - Best Performance/Cost ?
- Both require
  - Basis for comparison
  - Metric for evaluation

Many possible definitions of performance

- Every computer vendor will select one that makes them look good. How do you make sense of conflicting claims?

Introducing the 2.20 GHz Pentium®4 Processor

Built with Intel's 0.13 micron technology, the new 2.20 GHz Pentium® 4 processor delivers significant performance gains.



AMD

Q: *Why do end users need a new performance metric?*
A: End users who rely only on megahertz as an indicator for performance do not have a complete picture of PC processor performance and may pay the price of missed expectations.

Measuring Performance

- Latency (response time, execution time)
  - Minimize time to wait for a computation
- Energy/Power consumption
- Throughput (tasks completed per unit time, bandwidth)
  - Maximize work done in a given interval
  - = 1/latency when there is no overlap among tasks
  - > 1/latency when there is
    - In real processors there is always overlap (pipelining)
- All are important :
  - Architecture – Latency is important,
  - Embedded system – Power consumption is important,
  - Network – Throughput is important

Some Definitions

- Performance is in units of things/unit time
  - E.g., Hamburgers/hour
  - Bigger is better

- If we are primarily concerned with response time
  - $$\text{Performance}(x) = \frac{1}{\text{execution\_time}(x)}$$

- Relative performance: "X is N times faster than Y"

$$N = \frac{\text{Performance}(X)}{\text{Performance}(Y)} = \frac{\text{execution\_time}(Y)}{\text{execution\_time}(X)}$$

From previous slide

- Since execution time is the reciprocal of performance, the following relationship

holds:

$$n = \frac{\text{Execution time}_Y}{\text{Execution time}_X} = \frac{\dfrac{1}{\text{Performance}_Y}}{\dfrac{1}{\text{Performance}_X}} = \frac{\text{Performance}_X}{\text{Performance}_Y}$$

Relative Performance

"X is *n* times faster than Y" means:

$$\frac{\text{Execution time}_Y}{\text{Execution time}_X} = n$$

"X is *m*% faster than Y" means:

$$\frac{\text{Execution time}_Y}{\text{Execution time}_X} \times 100\% = m$$

Two notions of performance

Plane	DC to Paris	Speed	Passengers	Throughput (pmph)
747	6.5 hours	610 mph	470	286,700
Concorde	3 hours	1350 mph	132	178,200

- Which has higher performance?
  - Depends on the metric
    - Time to do the task (Execution Time, Latency, Response Time)
    - Tasks per unit time (Throughput, Bandwidth)
  - Response time and throughput are often in opposition

Throughput

- "the throughput of X is 1.3 times higher than Y" signifies here
  - that the number of tasks completed per unit time on computer X is 1.3 times the number completed on Y

Execution Time

- execution time can be defined in different ways depending on what we count.
  - The most straightforward definition of time is called ***wall-clock time***.
- ***response time, or elapsed time****, which is the* ***latency*** *to complete a task, including*
  - disk accesses, memory accesses, input/output activities, operating system overhead—everything

Measuring, Reporting, and Summarizing Performance

- With multiprogramming, the processor works on another program
- while waiting for I/O and may not necessarily minimize the elapsed time of one program.
- need a term to consider this activity. ***CPU time recognizes*** this distinction
  - the time the processor is computing, *not includ*ing the time waiting for I/O or running other programs. (Clearly, the response time seen by the user is the elapsed time of the program, not the CPU time.)

Three Components of CPU Performance

$\text{CPU time}_{X,P} = \text{Instructions executed}_P * \text{CPI}_{X,P} * \text{Clock cycle time}_X$

Cycles Per Instruction

CPU Performance

- **The Fundamental Law**

$$\text{CPU time} = \frac{\text{seconds}}{\text{program}} = \frac{\text{instructions}}{\text{program}} \times \frac{\text{cycles}}{\text{instruction}} \times \frac{\text{seconds}}{\text{cycle}}$$

- Three components of CPU performance:
  - Instruction count
  - CPI
  - Clock cycle time

	Inst. Count	CPI	Clock
Program	X		
Compiler (Technology)	X	X	
Inst. Set Architecture	X	X	X
μArch (Organization)		X	X
Physical Design(Hardware)			X

CPI - Cycles per Instruction

Let Fi be the frequency of type I instructions in a program. Then,
Average CPI:

$$\mathrm{CPI} = \frac{\text{Total Cycle}}{\text{Total Instruction Count}}$$

$$= \sum_{i=1}^{n} \mathrm{CPI}_i \times F_i \quad \text{where} \quad F_i = \frac{\mathrm{IC}_i}{\text{Instruction Count}}$$

$$\text{CPU time} = \text{Cycle time} \times \sum_{i=1}^{n} (\mathrm{CPI}_i \times \mathrm{IC}_i)$$

Example:

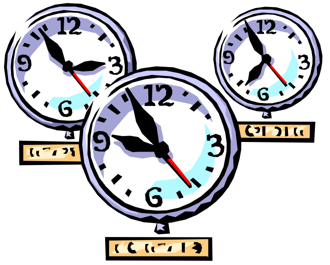
Instruction type	ALU	Load	Store	Branch
Frequency	43%	21%	12%	24%
Clock cycles	1	2	2	2

average CPI = 0.43 + 0.42 + 0.24 + 0.48 = 1.57 cycles/instruction

CPI

- The average number of clock cycles per instruction, or CPI, is a function of the machine and program.
  - The CPI depends on the actual instructions appearing in the program—a floating-point intensive application might have a higher CPI than an integer-based program.
  - It also depends on the CPU implementation. For example, a Pentium can execute the same instructions as an older 80486, but faster.
- It is common to each instruction took one cycle, making CPI = 1.
  - The CPI can be >1 due to memory stalls and slow instructions.
  - The CPI can be <1 on machines that execute more than 1 instruction per cycle (superscalar).

Clock cycle time



- One “cycle” is the minimum time it takes the CPU to do any work.
  - The clock cycle time or clock period is just the length of a cycle.
  - The clock rate, or frequency, is the reciprocal of the cycle time.
- Generally, a higher frequency is better.
- Some examples illustrate some typical frequencies.
  - A 500MHz processor has a cycle time of 2ns.
  - A 2GHz (2000MHz) CPU has a cycle time of just 0.5ns (500ps).

Execution time, again

$$\text{CPU time}_{X,P} = \text{Instructions executed}_P * \text{CPI}_{X,P} * \text{Clock cycle time}_X$$

- The easiest way to remember this is match up the units:

$$\frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} * \frac{\text{Clock cycles}}{\text{Instructions}} * \frac{\text{Seconds}}{\text{Clock cycle}}$$

- Make things faster by making any component smaller!!

	Program	Compiler	ISA	Organization	Technology
Instruction Executed					
CPI					
Clock Cycle TIme					

- Often easy to reduce one component by increasing another

Speedup Performance Laws
1) Amdahl's Law
(Module 3 of Syllabus)

- Quantitative Principles of Computer Design
  - Make common case fast
    - Improving frequent event rather than rare event improves overall performance
      - i.e in adding 2 numbers in CPU,
        - performance improved by optimizing common case of no overflow rather than rare case of flowchart

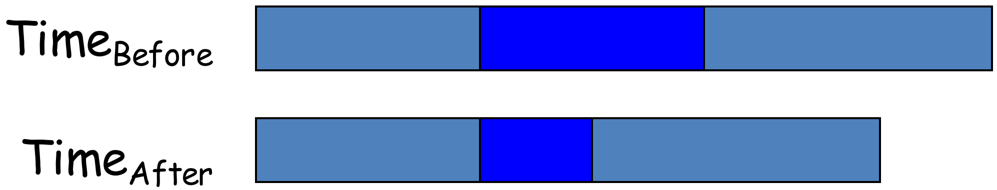
• **To find how much performance can be improved by making, a frequent event faster – Amdahl's law is used to quantify the values of enhancement (/ non-enhancement**)

Amdahl's Law

- A quick way to find speedup from an enhancement, which depends on:
  - Fraction of the computation time in the original machine that can be converted to take advantage of the enhancement
    - Example : If 20 seconds of execution time of a program that takes 60 seconds in total can use an enhancement, the fraction is **Fraction**$_{\textbf{enhanced}}$ = 20/60 is always less than 1
  - How much faster the task would run if the enhanced mode were used for the entire program
    - If the enhanced mode takes 10 seconds for some portion of program that can completely use enhanced mode and the original mode took 20 seconds for the same portion, the improvement is **Speedup**$_{\textbf{enhanced}}$ = 20/10 always greater than 1

Compute Speedup – Amdahl's Law

Speedup is due to enhancement(E):



$$\text{Speedup (E)} = \frac{\text{Execution time without E (Before)}}{\text{Execution time with E (After)}}$$

Suppose that enhancement E accelerates a fraction F (**Fraction**$_{\textbf{enhanced}}$) of the task by a factor S(**Speedup**$_{\textbf{enhanced}}$), and the remainder of the task is unaffected, what is the *Execution time*$_{after}$ and *Speedup(E)* ?

Amdahl's Law

$$\textit{Execution time}_{after} = ExTime_{before} \times \left[(1 - \mathbf{F}) + \frac{F}{S} \right]$$

$$\textit{Speedup(E)} = \frac{ExTime_{before}}{ExTime_{after}} = \frac{1}{\left[(1-F) + \frac{F}{S} \right]}$$

(F= fraction of total time that is speeded up ie 20 seconds out of total of 60 seconds = 20/60
20 secs is speeded up to 10 seconds,say)
S= 20/10

Amdahl's Law :Example Given in Previous slide

$Extime_{before} = 60$, **Fraction**$_{\textbf{enhanced}}$ (F) = 20/60, **Speedup**$_{\textbf{enhanced}}$ (S)= 20/10

Execution time$_{after}$ $= ExTime_{before} \times \left[(1-\mathbf{F}) + \dfrac{F}{S}\right]$ = 60 x 5/6=50

Execution time$_{after}$ = 60 - **60** x 20/60 + 60 x 20/60x 10/20
= 60-20 + 10 = 40 + 10 =50

ExecutionTimeNon Enhanced Portion + Execution time of Enhanced portion

$\left[(1-F) + \dfrac{F}{S}\right]$ = (1-20/60) + 20/60 x 10/20 = 2/3+1/6=5/6

(1-F) >>much greater than F/S i.e 2/3 >> 1/6

Speedup(E) $= \dfrac{ExTime_{before}}{ExTime_{after}} = \dfrac{6}{5}$

= 1.2

Check 60/50= 1.2

Corollary : We can't Speedup the task more than the reciprocal of 1 minus the fraction , that is the portion that is not speeded up , here (1/(1-20/60)=3/2=1.5)

Ref:pp 31 Patterson

Amdahl's Law – An Example

Q: Floating point instructions improved to run 2X;
but only 10% of execution time are FP ops. What is the execution time and speedup after improvement?

Ans:

$F = 0.1, S = 2$

$$ExTime_{after} = ExTime_{before} \times [\ (1\text{-}0.1) + 0.1/2\ ] = 0.95\ ExTime_{before}$$

$$Speedup = \frac{ExTime_{before}}{ExTime_{after}} = \frac{1}{0.95} = 1.053$$

<u>Amdahl's Law for Multiple Processors</u>

- **based on fixed workload or <u>fixed problem size</u>**
- computational workload *W is fixed while the number of processors that can work on W can be increased.*
- Denote the execution rate of *i processors as* $\boldsymbol{R}_i$ *then in a relative comparison they can be simplified as and* $R_1=1\ldots R_n=n$.
- *The workload is also simplified. We assume that the workload consists of sequential work Wα and n parallel work 1-αW ,*

where α is between 0 and 1. More specifically, this workload can be written in a vector form as, $(\alpha,0,...,0, 1-\alpha)W$, *or* $W_1=\alpha W$, $W_n=(1-\alpha)W$, $W_i=0$ *for all* $i \neq 1,n$.

Amdahl's Law for Multiple Processors

- The execution time of the given work by *n processors is then computed as,*

$$T_n = \frac{W_1}{R_1} + \frac{W_n}{R_n}$$

Speedup of *n processor system is defined using a ratio of execution time, i.e.,*

$$S_n = \frac{T_1}{T_n}$$

Amdahl's Law for Multiple Processors

Substituting the execution time in relation to *W gives*
:

$$S_n = \frac{W/1}{\frac{\alpha W}{1} + \frac{(1-\alpha)W}{n}} = \frac{n}{1+(n-1)\alpha}$$

$$S_n = \frac{n}{1+(n-1)\alpha}$$

Equation called Amdahl's Law

If the number of processors is increased infinity, the speedup becomes

$$S_\infty = \frac{1}{\alpha}$$

This is *sequential bottle neck* of multiprocessor system

The **speedup can NOT be increased to infinity** even **if the number of processors is increased to infinity**.

Gustafson's Law

- **For scaled up problems(** problem scaled to match the computing power of the machine as number of processors is increased)
- workload is scaled up to maintain a fixed execution time as the **number of processors increases,**

– workload is scaled up on an n-node machine as

$$W' = \alpha W + (1-\alpha)nW$$

Gustafson's Law

- Speedup for the scaled up workload is then,

$$S'_n = \frac{Single\ Processor Exeution Time}{n-Processor Execution Time}$$

$$S'_n = \frac{(\alpha W + (1-\alpha)nW)/1}{\frac{\alpha W}{1} + \frac{(1-\alpha)nW}{n}}$$

Simplifying above equation produces the Gustafson's law:

$$S'_n = \alpha + (1-\alpha)n$$

the speedup increases linearly.

- What Gustafson's law says
  - true **parallel power** of a large multiprocessor system is only **achievable when a large parallel problem is applied.**

MIPS and MFLOPS

- **MIPS:** millions of instructions per second:
  - MIPS = Inst. count/ (CPU time * 10**6) = Clock rate/(CPI*10^6)
  - easy to understand and to market
  - inst. set dependent, cannot be used across machines.
  - program dependent
  - <u>can vary inversely to performance! (why? read the book)</u>
- **MFLOPS:** million of FP ops per second.
  - less compiler dependent than MIPS.
  - not all FP ops are implemented in h/w on all machines.
  - not all FP ops have same latencies.
  - normalized MFLOPS: uses an equivalence table to even out the various latencies of FP ops.