



# Performance

---



- Today we'll discuss issues related to performance:
  - Latency/Response Time/Execution Time vs. Throughput
  - How do you make a reasonable performance comparison?
  - The 3 components of CPU performance
  - The 2 laws of performance

# 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 ✓

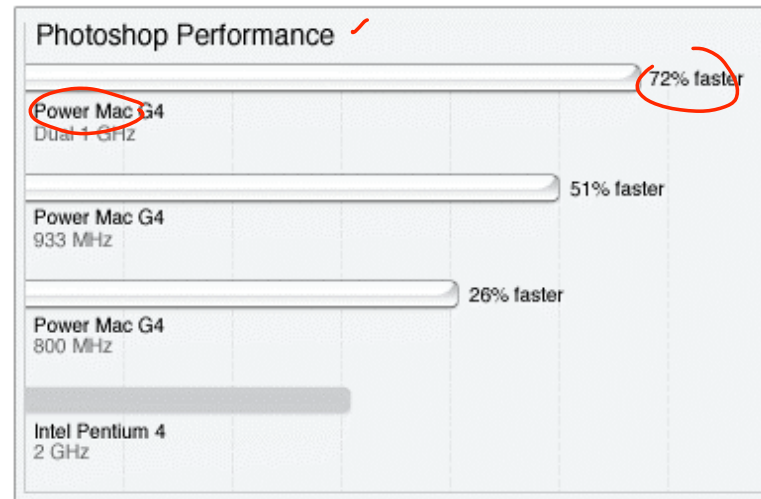
# Many possible definitions of performance

- 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.



**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.

## Two notions of performance

---

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

- 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

## 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
  - Performance(x) =  $\frac{1}{\text{execution\_time}(x)}$
- Relative performance: “X is N times faster than Y”

exe time ↑ ⇒ perform ↓

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

## Some Examples

---

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

- Time of Concorde vs. 747?

*Concorde*

- Throughput of Concorde vs. 747?

*747*

## Basis of Comparison

- When comparing systems, need to fix the workload
  - Which workload?

Workload	Pros	Cons
Actual Target Workload	Representative <u>                    </u>	Very specific ✓ Non-portable ✓ Difficult to run/measure ✓
Full Application Benchmarks	Portable ✓ Widely used ✓ Realistic ✓	<u>Less representative</u>
Small “Kernel” or “Synthetic” Benchmarks	Easy to run ✓ Useful early in design ✓	Easy to “fool” ✓
<u>Microbenchmarks</u>	Identify peak capability ✓ and potential bottlenecks	Real application performance may be much below peak

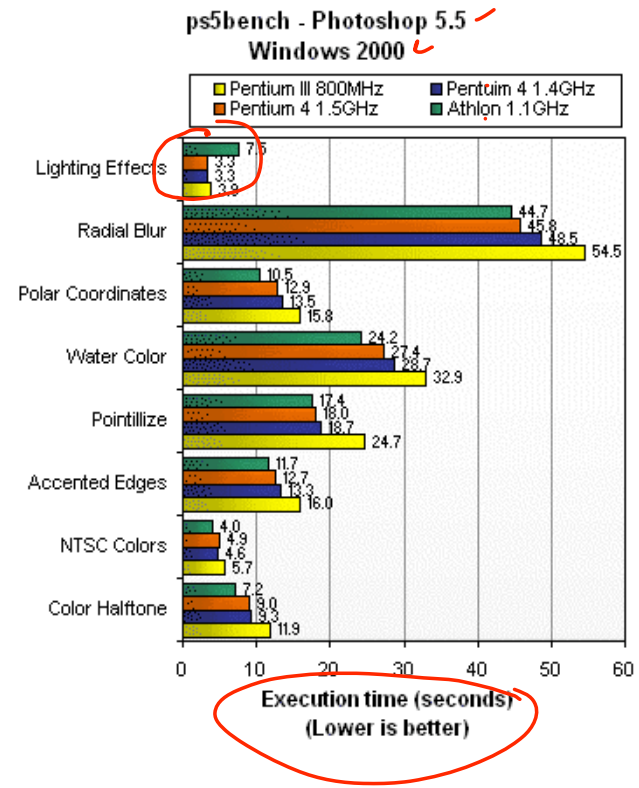


# Benchmarking

- Some common benchmarks include:
  - [Adobe Photoshop](#) for image processing ✓
  - [BAPCo SYSmark](#) for office applications ✓
  - [Unreal Tournament 2003](#) for 3D games ✓
  - [SPEC2000](#) for CPU performance

- The best way to see how a system performs for a variety of programs is to just show the execution times of all of the programs.
- Here are execution times for several different Photoshop 5.5 tasks, from

<http://www.tech-report.com>



# Summarizing performance

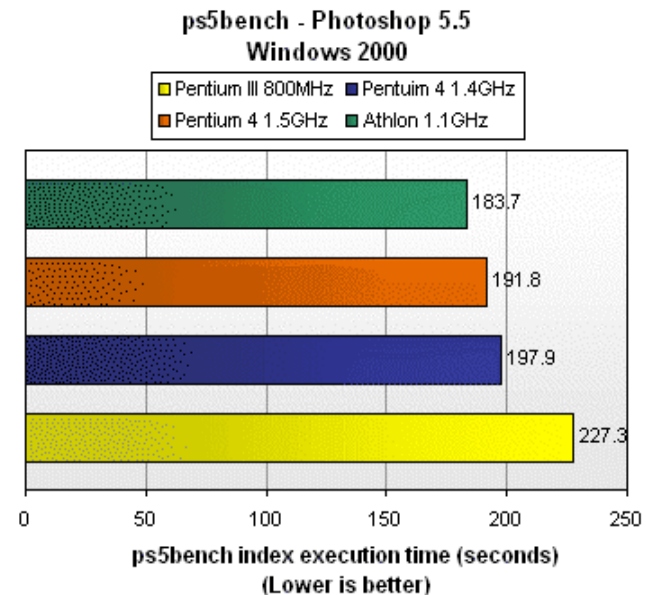
- Summarizing performance with a single number can be misleading—just like summarizing four years of school with a single GPA!
- If you must have a single number, you could **sum** the execution times.

This example graph displays the total execution time of the individual tests from the previous page.

- A similar option is to find the **average** of all the execution times.

For example, the 800MHz Pentium III (in yellow) needed 227.3 seconds to run 21 programs, so its average execution time is  $227.3/21 = 10.82$  seconds.

- A **weighted** sum or average is also possible, and lets you emphasize some benchmarks more than others.



# The components of execution time

- Execution time can be divided into two parts.
  - User time** is spent running the application program itself.
  - System time** is when the application calls operating system code.
- The distinction between user and system time is not always clear, especially under different operating systems.
- The Unix `time` command shows both.

```
salary.125 > time distill 05-examples.ps
Distilling 05-examples.ps (449,119 bytes)
10.8 seconds (0:11)
449,119 bytes PS => 94,999 bytes PDF (21%)
10.61u 0.98s 0:15.15 76.5%
```

User time                      System time                      “Wall clock” time (including other processes)

CPU usage = (User + System) / Total

# Three Components of CPU Performance

---

$$\text{CPU time}_{x,p} = \text{Instructions executed}_p * \text{CPI}_{x,p} * \text{Clock cycle time}_x$$

$\frac{1}{\text{freq}}$   
Cycles Per Instruction



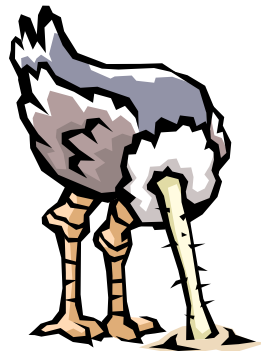
1GHz clock freq  
cycle = 1ns

# Instructions Executed

- Instructions executed:
  - We are not interested in the **static instruction count**, or how many lines of code are in a program.
  - Instead we care about the **dynamic instruction count**, or how many instructions are actually executed when the program runs.
- There are three lines of code below, but the number of instructions executed would be XXXX?.

```
Ostrich: | li    $a0, 1000  
          | sub   $a0, $a0, 1  
          | bne  $a0, $0, Ostrich
```

) 1000x



2001 dynamic inst

# CPI

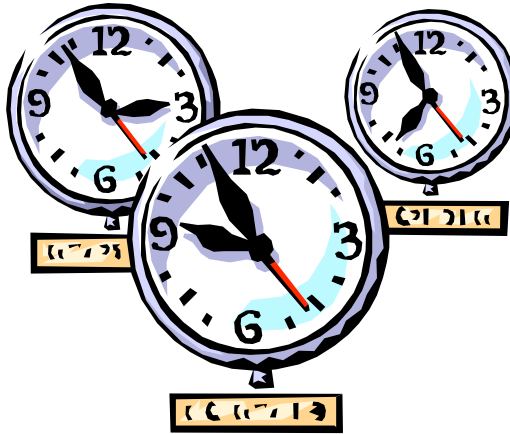
---

- 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).

↳ Alpha 20164  
PIII

# 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:

$$\frac{\text{Seconds}}{\text{Program}} = \frac{\cancel{\text{Instructions}}}{\text{Program}} * \frac{\overset{\text{CPI}}{\cancel{\text{Clock cycles}}}}{\cancel{\text{Instructions}}} * \frac{\overset{\text{Period}}{\text{Seconds}}}{\cancel{\text{Clock cycle}}}$$

- Make things faster by making any component smaller!!

	Program	Compiler	ISA	Organization	Technology
Instruction Executed	X	X	X		
CPI	X	X	X	X	~X
Clock Cycle Time			~X	X	X

- Often easy to reduce one component by increasing another



## Example 1: ISA-compatible processors

- Let's compare the performances two 8086-based processors.
  - An 800MHz AMD Duron, with a CPI of 1.2 for an MP3 compressor.
  - A 1GHz Pentium III with a CPI of 1.5 for the same program.
- Compatible processors implement identical instruction sets and will use the same executable files, with the same number of instructions.
- But they implement the ISA differently, which leads to different CPIs.

$$\begin{aligned}
 \text{CPU time}_{\text{AMD,P}} &= \text{Instructions}_p * \text{CPI}_{\text{AMD,P}} * \text{Cycle time}_{\text{AMD}} \\
 &= \textcircled{I} \cdot 1.2 \cdot \frac{1}{800 \times 10^6} = \text{CPU}_{\text{Duron}}
 \end{aligned}$$

$$\begin{aligned}
 \text{CPU time}_{\text{P3,P}} &= \text{Instructions}_p * \text{CPI}_{\text{P3,P}} * \text{Cycle time}_{\text{P3}} \\
 &= \textcircled{I} \cdot 1.5 \cdot \frac{1}{1000 \times 10^6} = \text{CPU}_{\text{Pentium III}}
 \end{aligned}$$

## Example 2: Comparing across ISAs

---

- Intel's Itanium (IA-64) ISA is designed facilitate executing multiple instructions per cycle. If an Itanium processor achieves an average CPI of .3 (3 instructions per cycle), how much faster is it than a Pentium4 (which uses the x86 ISA) with an average CPI of 1?
  - a) Itanium is three times faster
  - b) Itanium is one third as fast
  - c) Not enough information

# Improving CPI

---

- Many processor design techniques we'll see improve CPI
  - Often they only improve CPI for certain types of instructions

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

- $F_i$  = Fraction of instructions of type  $i$

- First Law of Performance:

Make the common case **fast**

## Example: CPI improvements

$$50\% \times 50\% = 25\%$$

- Base Machine:

Op Type	Freq (fi)	Cycles	CPIi
ALU	50%	3	
Load	20%	5	
Store	10%	3	
Branch	20%	2	

- How much faster would the machine be if:
  - we added a cache to reduce average load time to 3 cycles?
  - we added a branch predictor to reduce branch time by 1 cycle?
  - we could do two ALU operations in parallel?

# Amdahl's Law

---

- **Amdahl's Law** states that optimizations are limited in their effectiveness.

$$\text{Execution time after improvement} = \frac{\text{Time affected by improvement}}{\text{Amount of improvement}} + \text{Time unaffected by improvement}$$

- For example, doubling the speed of floating-point operations sounds like a great idea. But if only 10% of the program execution time  $T$  involves floating-point code, then the overall performance improves by just 5%.

$$\text{Execution time after improvement} = \frac{0.10 T}{2} + 0.90 T = 0.95 T$$

- What is the maximum speedup from improving floating point?
  - Second Law of Performance:

Make the fast case **common**

# Summary

---

- **Performance** is one of the most important criteria in judging systems.
- There are two main measurements of performance.
  - **Execution time** is what we'll focus on.
  - **Throughput** is important for servers and operating systems.
- Our main performance equation explains how performance depends on several factors related to both hardware and software.

$$\text{CPU time}_{x,p} = \text{Instructions executed}_p * \text{CPI}_{x,p} * \text{Clock cycle time}_x$$

- It can be hard to measure these factors in real life, but this is a useful guide for comparing systems and designs.
- **Amdahl's Law** tell us how much improvement we can expect from specific enhancements.
- The best **benchmarks** are real programs, which are more likely to reflect common instruction mixes.