Fundamentals of Computer Design

Rapid Pace of Development

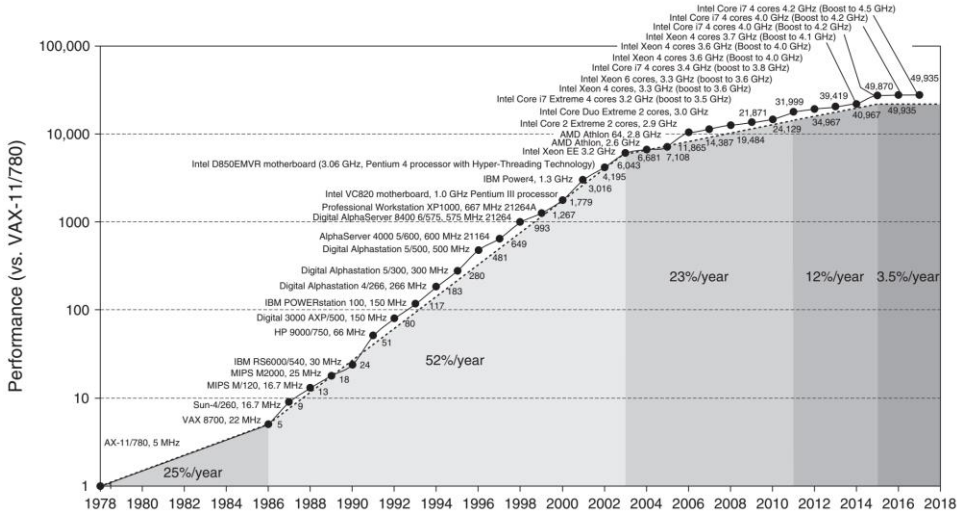
- IBM 7094 released in 1965
  - Featured interrupts
  - Could add floating point numbers at 350,000 instructions per second
  - Standard 32K of core memory in 36 bit words
  - Occupied an entire air conditioned room
  - System cost of about $3.5 million

This laptop

- Microsoft Surface Pro 5th Gen purchased in 2018
  - 13 GFLOPS (we'll see later this is not a particularly good benchmark)
  - 8GB of DRAM memory in 64 bit words
  - Occupies 12" diagonal, 1.7 pounds
  - System cost of about $1,300

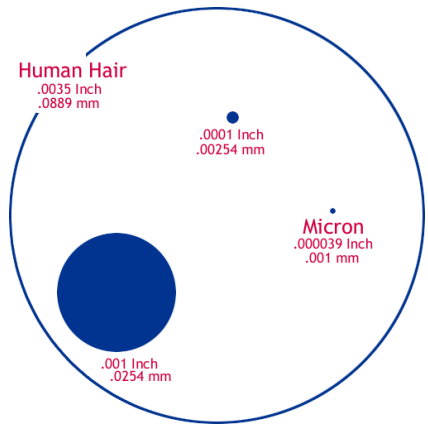
Moore's Law

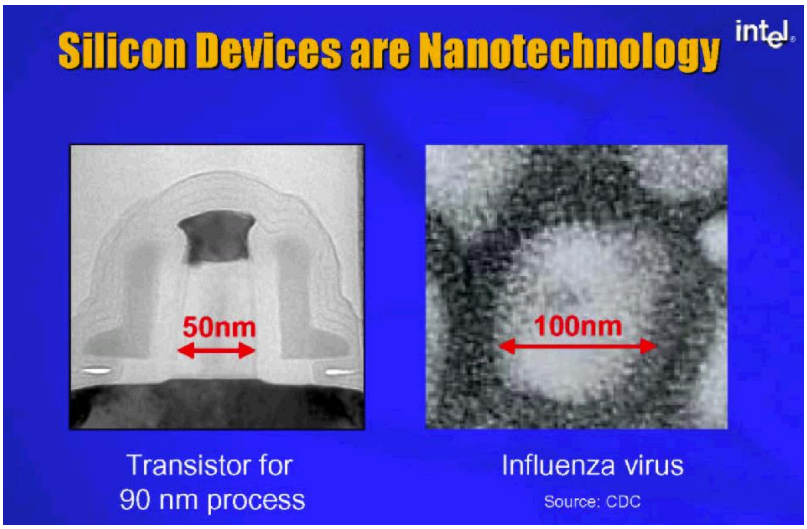
- Transistors per inch square
  - Twice as many after ~1.5-2 years
- Historical related trends
  - Processor performance
    Twice as fast after ~18 months
  - Memory capacity
    Twice as much in <2 years
- Not a true law but an observation
  - Are we getting close to hitting the physical limits?
  - Uniprocessor performance trends have peaked



The Shrinking Chip

- Human Hair: 100 microns wide
  - 1 micron is 1 millionth of a meter
- Bacterium: 5 microns
- Virus: 0.8 microns
- Early microprocessors: 10-15 micron technology
- 1997: 0.35 Micron
- 1998: 0.25 Micron
- 1999: 0.18 Micron
- 2001: 0.13 Micron
- 2003: 0.09 Micron
- 2007: 0.065 Micron
- 2009: 0.045 Micron
- 2014: 0.014 Micron (14 nm)
- 2018: 10 nm
- 2019: 7 nm
- 5 nm devices created in lab
- Single molecule device created at 0.17 nm should be the ultimate physical limit





7

RISC vs. CISC

- Big debate in the 80's
- Ideas from RISC won
  - Although you don't think of it as RISC, today's Intel Architecture adopted many RISC ideas internally

Terms for a Computer Designer

- Instruction Set Architecture – The assembly instructions visible to a programmer
- Organization – Mostly transparent to the programmer, but high-level design aspects such as how much cache, replacement policies, cache mapping algorithm, bus structure, internal CPU design.
- Computer Architecture – We'll refer to this as instruction set design, organization of the hardware design, and the actual hardware itself.

ISA for the first half of the class

- Mostly a MIPS-like ISA
  - RISC-V, open source
  - 32 general purpose and 32 floating point registers
  - Load-store architecture
- Memory Addressing
  - Byte addressing
  - Objects in RISC-V must be byte aligned
- Addressing Modes
  - Register, Immediate, Displacement

ISA for this class

- Types and sizes of operands
  - 8-bit ASCII to 64 bit double precision
- Operations
  - Data transfer, arithmetic, logical, control, floating point
- Control Flow instructions
- Encoding on an ISA
  - Fixed length vs. Variable length

Other Design Factors

Market Forces
Utilities, User Applications
Operating System, Programming Lang
Instruction Set Architecture
CPU, Memory, Interconnect, Buses
Power, Cooling, Logic, Fabrication

Conflicting Requirements

- To minimize cost implies a simple design
- To maximize performance implies complex design or sophisticated technology
- Time to market matters! Implies simple design and great secrecy
- Time to productivity! Implies need complete vertical solutions in place
- Don't mess up – requires simulation, QA, quantification

Technology Trends

- In 1943, Thomas Watson predicted "I think there is a world market for maybe five computers."
  - (The IBM PC was an "undercover" project and saved the company)
- In the 70's and 80's IBM pursued the high-speed Josephson Junction, spending $2 billion, before scrapping it and using CMOS like everyone else.

Current Trends in Architecture

- Cannot continue to leverage Instruction-Level parallelism (ILP)
  - Single processor performance improvement ended in 2003
- New models for performance:
  - Data-level parallelism (DLP)
  - Thread-level parallelism (TLP)
  - Request-level parallelism (RLP)
- These require explicit restructuring of the application

Classes of Computers

- Personal Mobile Device (PMD)
  - e.g. smart phones, tablet computers
  - Emphasis on energy efficiency and real-time
- Desktop Computing
  - Emphasis on price-performance
- Servers
  - Emphasis on availability, scalability, throughput
- Clusters / Warehouse Scale Computers
  - Used for "Software as a Service (SaaS)"
  - Emphasis on availability and price-performance
  - Sub-class: Supercomputers, emphasis: floating-point performance and fast internal networks
- Embedded Computers
  - Emphasis: price

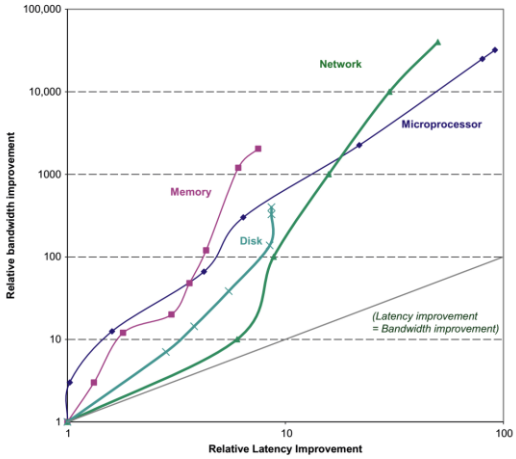
Flynn's Taxonomy

- Single instruction stream, single data stream (SISD)
- Single instruction stream, multiple data streams (SIMD)
  - Vector architectures
  - Multimedia extensions
  - Graphics processor units
- Multiple instruction streams, single data stream (MISD)
  - No commercial implementation
- Multiple instruction streams, multiple data streams (MIMD)
  - Tightly-coupled MIMD
  - Loosely-coupled MIMD

Trends in Technology

- Integrated circuit technology
  - Transistor density: 35%/year
  - Die size: 10-20%/year
  - Integration overall: 40-55%/year
- DRAM capacity: 25-40%/year (slowing)
  - 16gigabit DRAM in 2019
- Flash capacity: 50-60%/year
  - 10X cheaper/bit than DRAM
- Magnetic disk technology: 30%/year now 5%
  - 15-25X cheaper/bit then Flash
  - 300-500X cheaper/bit than DRAM

Bandwidth and Latency



Log-log plot of bandwidth and latency milestones

Transistors and Wires

- Feature size
  - Minimum size of transistor or wire in x or y dimension
  - 10 microns in 1971 to .032 microns in 2011
  - Transistor performance scales linearly
    - Wire delay does not improve with feature size! Function of resistance and capacitance
  - Integration density scales quadratically

Power and Energy

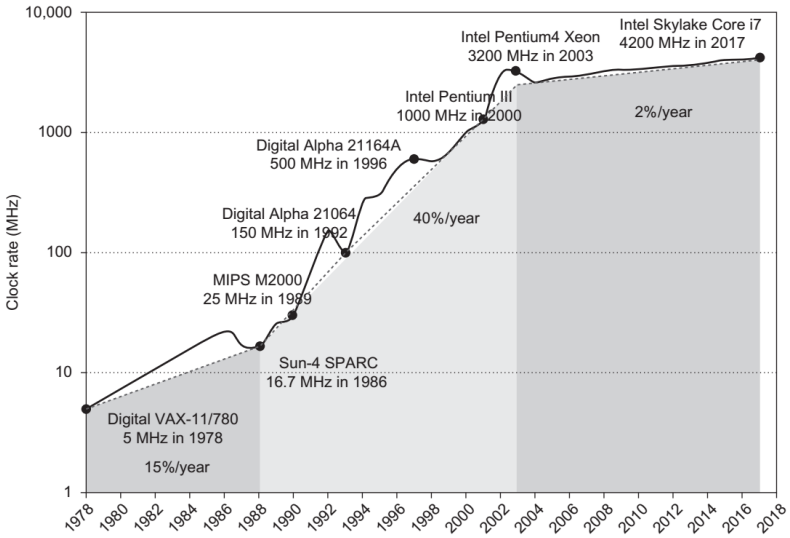
- Problem: Get power in, get power out
- Thermal Design Power (TDP)
  - Characterizes sustained power consumption
  - Used as target for power supply and cooling system
  - Lower than peak power, higher than average power consumption
- Clock rate can be reduced dynamically to limit power consumption
- Energy per task is often a better measurement

Dynamic Energy and Power

- Dynamic energy
  - Transistor switch from 0 -> 1 or 1 -> 0
  - ½ x Capacitive load x Voltage^2
- Dynamic power
  - ½ x Capacitive load x Voltage^2 x Frequency switched
- Reducing clock rate reduces power, not energy

- Intel 80386 consumed ~ 2 W
- 3.3 GHz Intel Core i7 consumes 130 W
- Heat must be dissipated from 1.5 x 1.5 cm chip
- This is the limit of what can be cooled by air

Clock Rate



Reducing Power

- Techniques for reducing power:
  - Do nothing well
  - Dynamic Voltage-Frequency Scaling
  - Low power state for DRAM, disks
    - Designing for the typical case
  - Overclocking, turning off cores

Trends in Cost

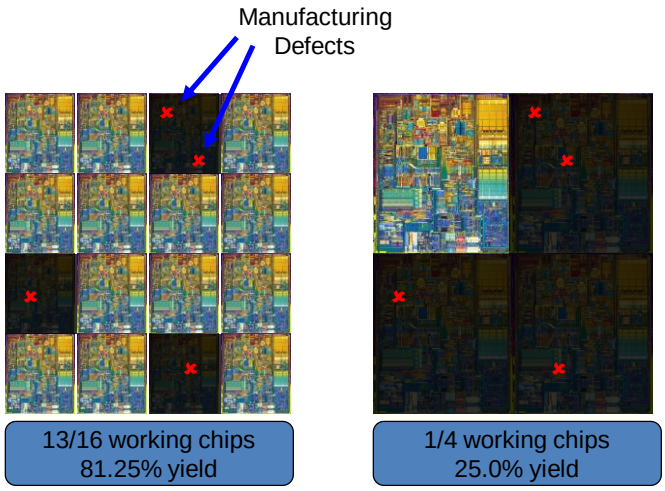
- Cost driven down by learning curve
  - Yield
- DRAM: price closely tracks cost
- Microprocessors: price depends on volume
  - 10% less for each doubling of volume

Chip Production

- Ingot of purified silicon – 1 meter long, sliced into thin wafers
- Chips are etched – much like photography
  - UV light through multiple masks
  - Circuits laid down through mask
- Process takes about 3 months

View of Cross-Section

Yield



Size Matters! (of the die, anyway)

Costs

$$IC\_Cost = \frac{Cost\_Die + Cost\_Test + Cost\_Packaging}{Yield}$$

$$Cost\_Die = \frac{Cost\_Wafer}{Dies\_per\_Wafer \times Die\_Yield}$$

$$Dies\_per\_Wafer = \frac{\pi(Wafer\_Radius)^2}{Die\_Area} - \frac{\pi(Wafer\_Diameter)}{\sqrt{2(Die\_Area)}} - Num\_Test\_Dies$$

$$Die\_Yield = Wafer\_Yield \times \left(1 + \frac{Defects\_per\_unit\_area \times Die\_Area}{\alpha}\right)^{-\alpha}$$

$0.02/cm^2$ in 2010

Complexity of manufacturing; a typical $\alpha = 4$

Cost per die

- The book has some examples of computing the die yield for various wafer and die sizes.
- For a designer, the wafer yield, cost, and defects are all determined by the manufacturing process.
- The designer has control over the die area.
  - If alpha is typically 4, then this means that the die_yield is proportional to (Die_Area)$^{-4}$. Plugging this in to the equation for die cost gives us:

$$Die\_Cost = f(Die\_Area^4)$$

- Size matters!

Optimization Problem

- Optimize price/performance ratio
- Conflicting goals due to interactions between components
  - If adding a new feature, die size goes up, number of defects goes up and fewer dies per wafer, may need more testing, power usage may increase requiring larger battery or heat sink, etc…

Measuring Performance

- What does it mean to say one computer is faster than another?
  - Historically, advertisers and consumers have used clock speed. 2Ghz vs. 1Ghz means the first is twice as fast as the second?
  - Pentium 4E in 2004
    - Whopping 3.6 Ghz!
    - 31 stage integer pipeline!
  - Core i5 7200U in 2018
    - Normally 2.5Ghz, up to 3.1Ghz

Execution Time – A Better Metric

- Compare two machines, X and Y, on the same task:

$$n = \frac{Execution - Time(Y)}{Execution - Time(X)}$$

- Another metric sometimes used is performance, which is just the reciprocal of execution time:

$$Performance(X) = \frac{1}{Execution - Time(X)}$$

Measuring Execution Time

- More difficult than it seems to measure execution time
  - *Wall-clock time, response time,* or *elapsed time* - Total time to complete a task. Note the system might be waiting on I/O, performing multiple tasks, OS activities, etc.
  - *CPU time* – Only the time spent computing in the CPU, not waiting for I/O. This includes time spent in the OS and time spent in user processes.
  - *System CPU Time* – Portion of CPU time spent performing OS related tasks.
  - *User CPU Time* – Portion of CPU time spent performing user related tasks.

What programs should we run?

- Real Programs – Run the programs users will use. Examples are compilers, office tools, etc.
  - Unfortunately there is a porting problem among different architectures, so it might not be a fair comparison if the same software is coded differently.
- Kernels – These are small, intensive, key pieces from real programs that are run repeatedly to evaluate performance.
  - Examples include Livermore Loops and Linpack. Here is a code fragment from Livermore Loops:

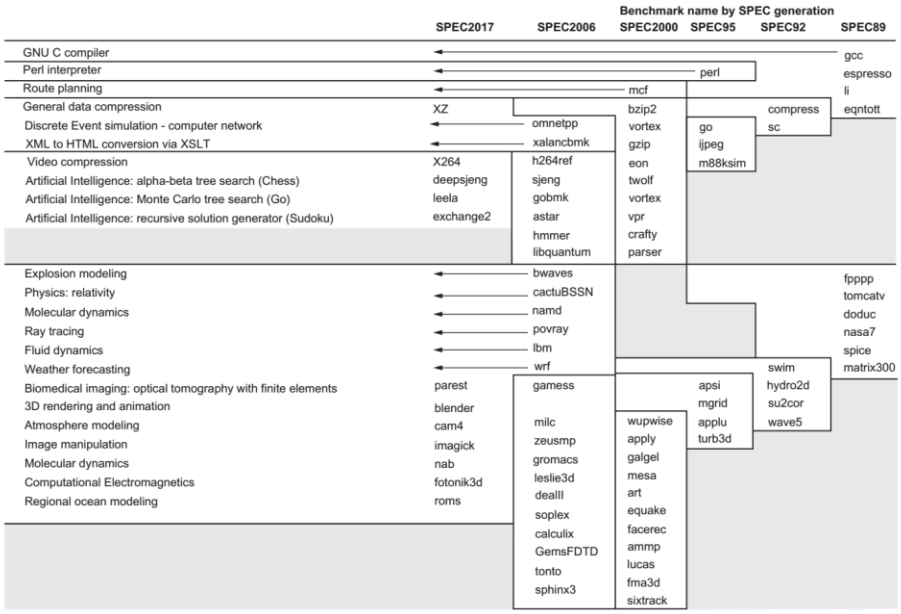
```
for (l=1; l<=loop; l++) {
          for (k=0; k<n; k++) {
            x[k] = q + y[k] * (r*z[k+10]+t*z[k+11]);
          }
}
```

Benchmarks

- Toy benchmarks - this includes small problems like quicksort, sieve of eratosthenes, etc.
  - They are best saved for 201 programming assignments!
- Synthetic benchmarks - These are similar to kernels, but try to match the average frequency of operations and operands of a large set or programs.
  - Examples: Whetstone and Dhrystone
  - The problem is no user really runs these either. Programs typically reside entirely in cache and don't test the entire system performance!

Benchmarks

- Benchmark Suites - These are a collection of benchmarks together in an attempt to measure the performance of processors with a variety of applications.
  - Suffers from problems of OS support, compiler quality, system components, etc.
  - Suites such as CrystalMark or the SPEC (www.spec.org) benchmark seem to be required for the industry today, even if the results may be somewhat meaningless.
  - https://www.spec.org/cpu2017/



Programs in SPECint92 and SPECfp92

Benchmark	Source	Lines of Code	Description
Espresso	C	13500	Minimize Boolean functions.
Li	C	7413	Lisp interpreter that solves 8 queens problem
Compress	C	1503	LZ compression on a 1Mb file
Gcc	C	83589	GNU C Compiler

Benchmark	Source	Lines of Code	Description
Spice2g6	FORTRAN	18476	Circuit simulation
Alvinn	C	272	Neural network training simulation
Ear	C	4483	Inner ear model that filters and detects various sounds
Su2cor	FORTRAN	2514	Compute masses of elementary particles from Quark-Gluon theory

Active 2017 SPEC Benchmarks

Category	Name	Measures performance of
Cloud	Cloud_IaaS 2016	Cloud using NoSQL database transaction and K-Means clustering using map/reduce
CPU	CPU2017	Compute-intensive integer and floating-point workloads
Graphics and workstation performance	SPECviewperf® 12	3D graphics in systems running OpenGL and Direct X
	SPECwpc V2.0	Workstations running professional apps under the Windows OS
	SPECapcSM for 3ds Max 2015™	3D graphics running the proprietary Autodesk 3ds Max 2015 app
	SPECapcSM for Maya® 2012	3D graphics running the proprietary Autodesk 3ds Max 2012 app
	SPECapcSM for PTC Creo 3.0	3D graphics running the proprietary PTC Creo 3.0 app
	SPECapcSM for Siemens NX 9.0 and 10.0	3D graphics running the proprietary Siemens NX 9.0 or 10.0 app
	SPECapcSM for SolidWorks 2015	3D graphics of systems running the proprietary SolidWorks 2015 CAD/CAM app
High performance computing	ACCEL	Accelerator and host CPU running parallel applications using OpenCL and OpenACC
	MPI2007	MPI-parallel, floating-point, compute-intensive programs running on clusters and SMPs
	OMP2012	Parallel apps running OpenMP
Java client/server	SPECjbb2015	Java servers
Power	SPECpower_ssj2008	Power of volume server class computers running SPECjbb2015
Solution File Server (SFS)	SFS2014	File server throughput and response time
	SPECsfs2008	File servers utilizing the NFSv3 and CIFS protocols
Virtualization	SPECvirt_sc2013	Datacenter servers used in virtualized server consolidation

Benchmark Problems

- Benchmark mistakes
  - Only average behavior represented in test workload
  - Loading level controlled inappropriately
  - Caching effects ignored
  - Ignoring monitoring overhead
  - Not ensuring same initial conditions
  - Collecting too much data but doing too little analysis
- Benchmark tricks
  - Compiler (soft)wired to optimize the workload
  - Very small benchmarks used
  - Benchmarks manually translated to optimize performance

Benchmark Trick Example

- Certain flags during compilation can have a huge effect on final execution time for some tests
  - The Whetstone loop contains the following expression:
    - SQRT(EXP(X))
  - A brief analysis yields:

$$\sqrt{e^x} = e^{x/2} = EXP(X/2)$$

- It would be surprising to see such an optimization automatically performed by a compiler due to the expected rarity of encountering SQRT(EXP(X)). Nevertheless, several compilers did perform this optimization!

Comparing Performance

- The table lists the time in seconds that each computer requires to run a particular program.

	Computer A	Computer B	Computer C
Program P1	1	10	20
Program P2	1000	100	20
Total Time	1001	110	40

Which computer is better? For P1, A is 10 times faster than B, but for P2 B is 10 times faster than A.

Total Execution Time
A consistent summary measure

- Simplest approach: compare relative performance in total execution time
- So then B is 9.1 times faster than A (1001/110), while C is 2.75 times faster than B. Of course this is all relative to the programs that are selected.

Other Total Execution Time Metrics

- Arithmetic mean $\frac{1}{n}\sum_{i=1}^{n} Time_i$
- Weighted arithmetic mean $\sum_{i=1}^{n} Weight_i \times Time_i$
- Normalize the execution time with respect to some reference machine.
  - Geometric mean $\left(\prod_{i=1}^{n} ExecutionTimeRatio_i\right)^{\frac{1}{n}}$

Geometric Mean

- For the previous example referenced to A:
- For A $\left(\frac{1}{1} * \frac{1000}{1000}\right)^{\frac{1}{2}} = 1$
- For B $\left(\frac{10}{1} * \frac{100}{1000}\right)^{\frac{1}{2}} = 1$
- For C $\left(\frac{20}{1} * \frac{20}{1000}\right)^{\frac{1}{2}} = 0.63$

This says that the execution time of these programs on C is 0.63 of A and B.

Quantitative Principles of Computer Design

- Take Advantage of Parallelism
  - Instruction level, application level, system level
- Principle of Locality
  - Temporal and spatial
- Focus on the Common Case
  - In design, favor the frequent case over the infrequent case
    - E.g. if adding two numbers rarely results in overflow, can improve performance by optimizing the more common case of no overflow
      - Will be slower when overflow occurs, but rare so overall performance will be improved

Amdahl's Law

- The performance improvement to be gained from using some faster mode of execution is limited by the fraction of time the faster mode can be used.
- Speedup = (Performance for entire task using enhancement) / (Performance for entire task without enhancement)
- Another variant is based on the ratio of Execution times, where Execution time = 1/speedup.

Speedup

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

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

Fraction(enhanced) is just the fraction of time the enhancement can be used. For example, if a new enhancement is 20 times faster but can only be used 5% of the time:

$$OverallSpeedup = \frac{1}{(1-0.05) + \frac{0.05}{20}} = 1.049$$

Example

- Consider a critical benchmark. Floating Point Square Root is responsible for 20% of the execution time. You could increase this operation by a factor of 10 via hardware. Or at the same cost, you could make all FP instructions run 2 times faster, which accounts for 50% of the execution time. Which is better?

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

CPU Performance

- All commercial computers are synchronous – they have a clock that "ticks" once per cycle.
  - Usually the cycle time is a constant, but it may vary (e.g. SpeedStep).
  - Duration is usually on the order of nanoseconds, or more commonly the rate is expressed instead , e.g. 100 Mhz.
- CPU time for a program can then be expressed two ways:

$$\text{CPU time} = \text{CPU clock cycles for a program} \times \text{Clock cycle time}$$

$$\text{CPU time} = \frac{\text{CPU clock cycles for a program}}{\text{Clock rate}}$$

Cycles Per Instruction

- We can count the number of instructions executed – the instruction path length or Instruction Count (IC). Given IC and the number of clock cycles we can calculate the average number of clock cycles per instruction, or **CPI**:

$$\text{CPI} = \frac{\text{CPU clock cycles for a program}}{\text{Instruction count}}$$

- With a little algebraic manipulation we can use CPI to compute CPU Time:
  - CPU clock cycles for a program = CPI * IC

    Substitute this in for CPU Time..
  - CPU Time = CPI * IC * Clock Cycle Time

    Or
  - CPU Time = CPI * IC / ClockRate

CPI

- Unfortunately, CPI, IC, and the Clock Cycle Time are all subtly inter-related.
  - CPI and cycle time depend on the instruction set
  - The instruction set depends on the hardware
  - The hardware impacts the cycle time, etc.
- Sometimes CPI is more useful to deal with in terms of all individual instructions. We can denote this by:
  - CPU Time = $(\sum_{i=1}^{n} CPI_i \times IC_i) \times ClockCycleTime$

  and

  CPI Total= $(\sum_{i=1}^{n} CPI_i \times \frac{IC_i}{InstructionCount})$

CPI Example

- CPI is useful because it is measurable, unlike the nebulous "fraction of execution" in Amdahl's equation. Consider the previous example to compute the speedup of Floating Point Square Root:
- Suppose you run your program and as it runs collect the following data:
  - 100 total instructions
  - 25 of these instructions are floating point instructions
  - 2 of these instructions are floating point square root
  - 100 clock cycles were spent on floating point instructions
    - 40 of these 100 cycles were spent on floating point square root
  - 100 clock cycles were spent on non-floating point instructions

CPI Example

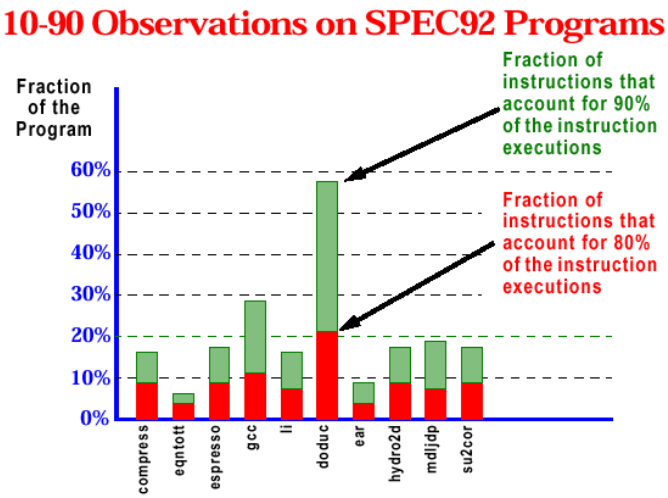
- Frequency of FP operations = 25/100 = 0.25
- Frequency of FPSQ operations = 2/100 = 0.02
- Ave CPI of FP operations = 100 / 25 = 4
- Ave CPI of FPSQ operations = 40 / 2 = 20
- Ave CPI of non-FP operations = 100 / 75 = 1.33

- If we could reduce the CPI of FPSQ by 10 (down to 2), or reduce the CPI of all FP operations by 2, which is better?
  - Will calculate in class

Measuring Components of CPU Performance

- Cycle Time is easy to measure for an existing CPU (whatever frequency it is running at).
- Cycle Time is hard to measure for a CPU in design! The logic design, VLSI process, simulations, and timing analysis need to be done.
- IC – This is one thing that is relatively easy to measure, just count up the instructions. This can be done with instruction trace, logging, or simulation tools.
- CPI – This can be difficult to measure exactly because it depends on the processor organization as well as the instruction stream. Pipelining and caching will affect the CPI, but it is possible to simulate the system in design and estimate the CPI.

Example on effect of locality



Common Fallacies, Pitfalls

- Falling prey to Amdahl's Law
  - Should measure speedup before spending effort to enhance it
- Single point of failure
  - E.g. single fan may limit reliability of the disk subsystem
- Fallacy: The cost of the processor dominates the cost of the system

Common Fallacies, Pitfalls

- Fallacy: Benchmarks remain valid indefinitely
  - Benchmark reverse engineering in compilers
- Fallacy: The rated mean time to failure of X is 1,200,000 hours or almost 140 years, so X will practically never fail
  - More useful measure would be % of disks that fail
- Fallacy: Peak performance tracks observed performance
  - Example: Alpha in 1994 announced as being capable of executing 1.2 billion instructions per second at its 300 Mhz clock rate.

Common Fallacies, Pitfalls

- Fault detection can lower availability
  - Hardware has a lot of state that is not always critical to proper operation
    - E.g. if cache prefetch fails, program will still work, but a fault detection mechanism could crash the program or raise errors that take time to process
    - Less than 30% of operations on the critical path for Itanium 2

Common Fallacies, Pitfalls

- Fallacy: MIPS (Millions of Instructions Per Second) is a useful benchmark (along with Ghz)
- Example: Say multiply FP instruction requires 4 clock cycles. However, instead of executing the FP multiply instruction, we could instead use a software floating point routine that used only Integer instructions.
- Since the integer instructions are simpler, they will require fewer clock cycles. For simplicity say that each integer instruction requires 2 clock cycles and it takes 20 of them to implement the FP multiply. Then for a 1 Mhz machine:
  - FP Multiply has a CPI of 4
  - MIPS = 1 / 4 = 0.25
- The software FP Multiply using integer instructions has a CPI of 2
  - MIPS = 1 / 2 = 0.50
- The software version has higher MIPS! Lost in the analysis is that it takes many more integer instructions than floating point instructions to do the same thing.