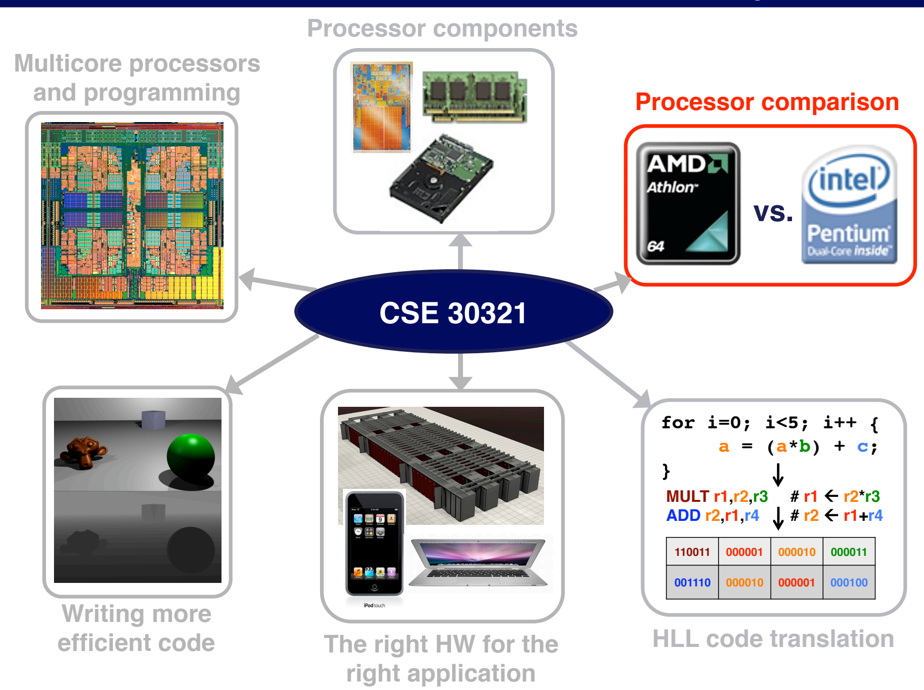
Lecture 04
Performance Metrics and Benchmarking

Recommended Readings

- Readings
  - H&P: Chapter 1



Which is “the best”?

Measuring & Improving Performance

(if planes were computers...)

Which is best?

Plane	People	Range (miles)	Speed (mph)	Avg. Cost (millions)
737-800	162	3,060	530	63.5
747-8I	467	8000	633	257.5
777-300	368	5995	622	222
787-8	230	8000	630	153

An “architecture” example

1 GHz clock rate, each instruction takes ~1.2 cycles to execute

How do we determine which machine is better?

2 GHz clock rate, each instruction takes ~1.8 cycles to execute

```
      ...
MOV R1, d(8)
Add R2, R3, R1
Sub R5, R2, R1
MOV d(9) R5
Add R4, R3, R0
      ...
```


May be a minimum performance requirement

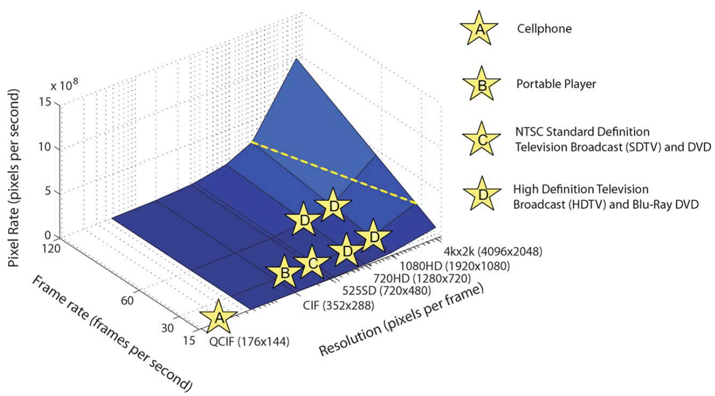


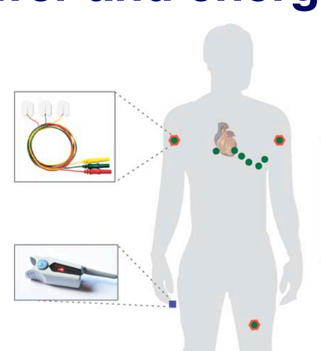
Fig. 1. *Performance requirements for various applications based on frame rate and resolution [6]. Yellow dashed line shows limit of H.264/AVC standard. Next-generation standard is expected to reach above this line.*

Technologies for Ultradynamic Voltage Scaling

By Anantha P. Chandrakasan, *Fellow IEEE*, Denis C. Daly, *Member IEEE*, Daniel Frederic Finchelstein, *Member IEEE*, Joyce Kwong, *Student Member IEEE*, Yogesh Kumar Ramadass, *Member IEEE*, Mahmut Ersin Sinangil, *Student Member IEEE*, Vivienne Sze, *Student Member IEEE*, and Naveen Verma, *Member IEEE*

Vol. 98, No. 2, February 2010 | Proceedings of the IEEE

Power and energy are important too



Monitoring	Sample Rate	# Cycles / Sample	Processor Frequency
Pulseoximetry	1kHz	331	331kHz
Single-lead ECG	200Hz	4990	1MHz
12-lead ECG	1kHz	25700	25.7MHz

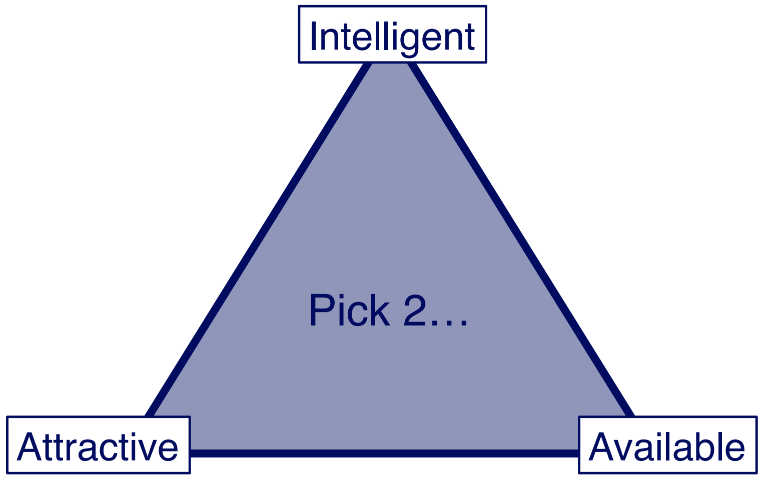
Fig. 2. *Scenarios for monitoring cardiac activity with varying real-time processing demands. For each application, locations of electrodes/probes on the body are shown, as well as the required clock frequency of the sensor processor. (Photos courtesy of GANFYD.)*

Technologies for Ultradynamic Voltage Scaling

By Anantha P. Chandrakasan, *Fellow IEEE*, Denis C. Daly, *Member IEEE*, Daniel Frederic Finchelstein, *Member IEEE*, Joyce Kwong, *Student Member IEEE*, Yogesh Kumar Ramadass, *Member IEEE*, Mahmut Ersin Sinangil, *Student Member IEEE*, Vivienne Sze, *Student Member IEEE*, and Naveen Verma, *Member IEEE*

Vol. 98, No. 2, February 2010 | Proceedings of the IEEE

Architecture: kinda like dating…

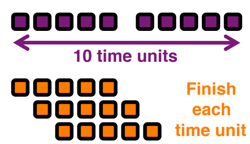


Characterizing Performance

- **How can one computer's performance be understood or two computers be compared?**
- **What factors go into achieving "good performance"?**
  - **Raw CPU speed?**
  - **Memory speed or bandwidth?**
  - **I/O speed or bandwidth?**
  - **The operating system's overhead?**
  - **The compiler?**
  - **Battery life?**
- **Critical to succinctly summarize performance, and meaningfully compare.**

Common (and good) performance metrics

- **latency: response time, execution time**
  - **good metric for fixed amount of work (minimize time)**
- **throughput: bandwidth, work per time, "performance"**
  - **= (1 / latency) when there is NO OVERLAP**
  - **> (1 / latency) when there is overlap**
    - **in real processors there is always overlap**
  - **good metric for fixed amount of time (maximize work)**



- **comparing performance**
  - **A is N times faster than B if and only if:**
    - perf(A)/perf(B) = **time(B)/time(A) = N**
  - **A is X% faster than B if and only if:**
    - perf(A)/perf(B) = **time(B)/time(A) = 1 + X/100**

Throughput vs. Latency

- **What is better?**
  - **A machine that takes 1 ns to do "task X" 1 time**
  - **A machine that takes 15 ns to do "task X" 30 times…**
    - **…but 5 ns to do "task X" 1 time**
  - **The 1st machine has a lower latency for a single operation…**
  - **The 2nd machine has better throughput for multiple operations**
  - **Which is better depends on what kind of computation you need to do**

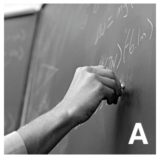
Execution time and throughput are really good performance metrics in that they're "lowest common denominators"

(i.e. if X finishes in 5 seconds and Y finishes in 10, its hard to make the case that Y is faster!)

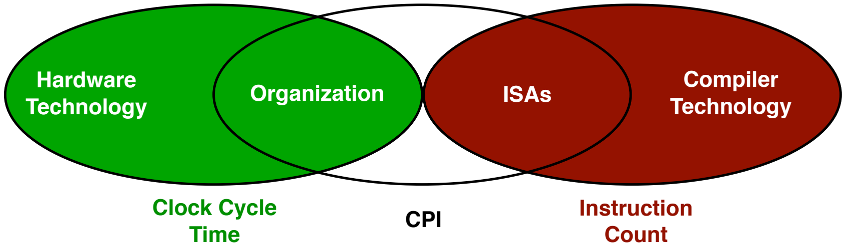
Later, we discuss a few other performance metrics that you may sometimes see - but are generally not as good and/or misleading.

A CPU : The Bigger Picture

$$\frac{Instructions}{Program} \times \frac{Clock\ cycles}{Instruction} \times \frac{Seconds}{Clock\ Cycle} = \frac{Seconds}{Program} = CPU\ time$$

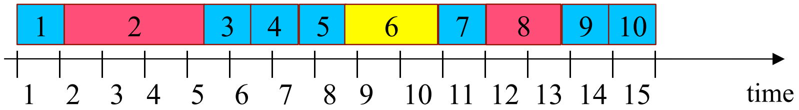


- We can see CPU performance dependent on:
  - Clock rate, CPI, and instruction count
- CPU time is directly proportional to all 3:
  - Therefore an x % improvement in any one variable leads to an x % improvement in CPU performance
- But, everything usually affects everything:



IC, CPI and IPC

Consider the following:



Total Execution Time	= 15 cycles
Instruction Count (IC) = Number of Instructions	= 10
Average number of cycles per instruction (CPI)	= 15/10 = 1.5
Instructions per Cycle (IPC)	= 10/15 = 0.66

Can CPI < 1?

Different Types of Instructions

- Multiplication takes more time than addition
- Floating point operations take longer than integer operations
- Memory accesses take more time than register accesses
- NOTE: changing the cycle time often affects the number of cycles an instruction will take

$$\text{CPU Clock Cycles} = \sum_{i=1}^{n} CPI_i * IC_i = AvgCPI * IC$$

Question: Measurement Comparison

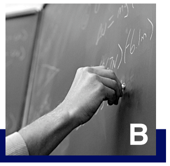
- **Given that two machines have the same ISA, which measurement is always the same for both machines running program P?**
  - **Clock Rate:**
  - **CPI:**
  - **Execution Time:**
  - **Number of Instructions:**
  - **MIPS:**

The Power of Compiler

A compiler designer is trying to decide between two code sequences for a particular machine. The machine supports three classes of instructions: A, B, and C, which take one, two, and three cycles (respectively):

Sequence 1 contains: 2 A's, 1 B, and 2 C's
Sequence 2 contains: 4 A's, 1 B, and 1 C

Which sequence is faster? By how much? What is the CPI of each?

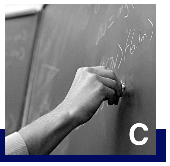


Metrics

- **Metrics Discussed:**
  - **Execution Time (instructions, cycles, seconds)**
  - **Machine Throughput (programs/second)**
  - **Cycles Per Instruction (CPI)**
  - **Instructions Per Cycle (IPC)**
- **Other Common Measures**
  - **MIPS (millions of instructions per second)**
  - **MFLOPS (megaflops) = millions of floating point operations per second**

Not all benchmarks are good…

- **Example: MIPS (millions of instructions per second)**
  - **(instruction count / execution time in seconds) x 10^{-6}**
  - **instruction count is not a reliable indicator of work**
    - **Prob #1: some optimizations add instructions**
    - **Prob #2: work per instruction varies**
      - **(FP mult >> register move)**
    - **Prob #3: ISAs not equal (3 Pentium instrs != 3 AMD instrs)**
      - **You'll see more when we talk about addressing modes**
        - » **Auto-increment may be a good example…**
  - **may vary inversely with actual performance**



Good Benchmarks: Real Programs

- real programs
  - (plus) only accurate way to characterize performance
  - (minus) requires considerable work (porting)
- Standard Performance Evaluation Corporation (SPEC)
  - http://www.spec.org
  - collects, standardizes and distributes benchmark suites
  - consortium made up of industry leaders
  - SPEC CPU (CPU intensive benchmarks)
    - SPEC89, SPEC92, SPEC95, SPEC2000, SPEC2006
  - other benchmark suites
    - SPECjvm, SPECmail, SPECweb
- Other benchmark suite examples: TPC-C, TPC-H for databases

SPEC CPU 2000

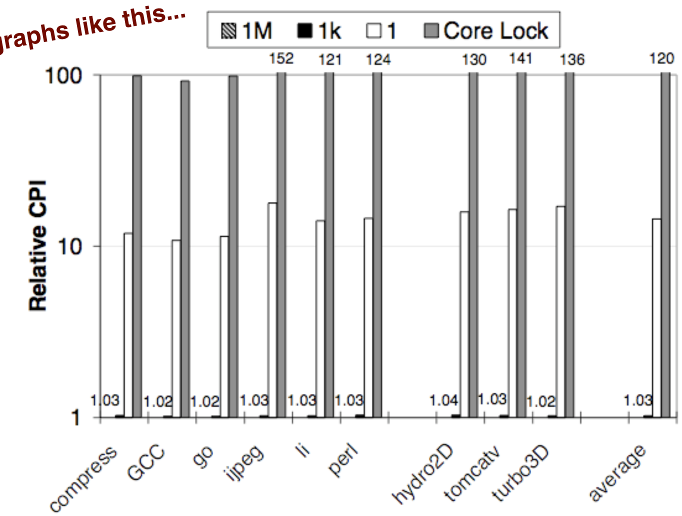
- 12 integer programs (C, C++)
  - gcc (compiler), perl (interpreter), vortex (database)
  - bzip2, gzip (replace compress), crafty (chess, replaces go)
  - eon (rendering), gap (group theoretic enumerations)
  - twolf, vpr (FPGA place and route)
  - parser (grammar checker), mcf (network optimization)
- 14 floating point programs (C, FORTRAN)
  - swim (shallow water model), mgrid (multigrid field solver)
  - applu (partial diffeq's), apsi (air pollution simulation)
  - wupwise (quantum chromodynamics), mesa (OpenGL library)
  - art (neural network image recognition), equake (wave propagation)
  - fma3d (crash simulation), sixtrack (accelerator design)
  - lucas (primality testing), galgel (fluid dynamics), ammp (chemistry)

SPEC 2000

- Different programs in the suite stress different parts of the architecture
  - For example:
    - One benchmark may be memory intensive…
    - …another may be compute intensive…
    - …another may be I/O intensive…
  - Ideally, show wins on all aspects, but most often not the case - or the point

SPEC 2000 (and architecture evaluation)

Often see graphs like this…



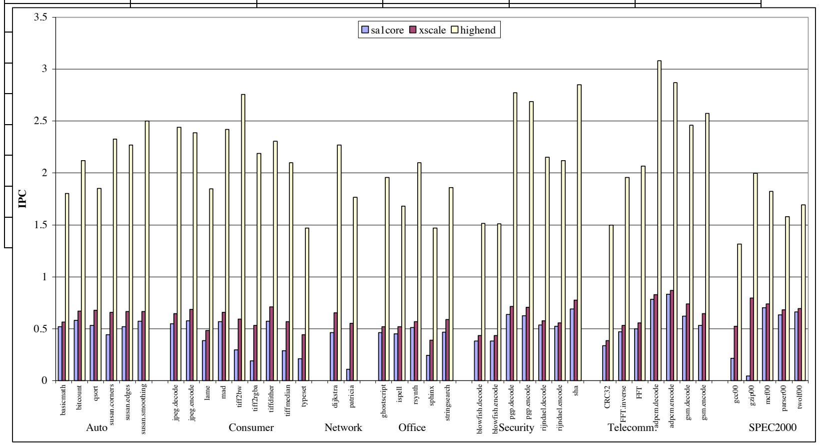
(and interestingly, now such a graph without accompanying power analysis is viewed as incomplete)

Other suites

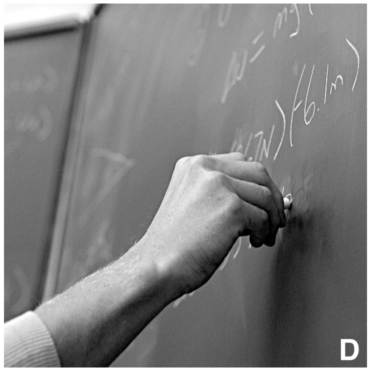
MiBench: A free, commercially representative embedded benchmark suite

Table 1: MiBench Benchmarks

Auto./Industrial	Consumer	Office	Network	Security	Telecomm.



Some additional examples



Amdahl's Law

- **Qualifies performance gain**
- **Amdahl's Law defined…**
  - **The performance improvement to be gained from using some faster mode of execution is limited by the amount of time the enhancement is actually used.**
- **Amdahl's Law defines speedup:**

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

Amdahl's Law and Speedup

- **Speedup tells us how much faster the machine will run with an enhancement**
- **2 things to consider:**
  - **1st…**
    - **Fraction of the computation time in the original machine that can use the enhancement**
    - **i.e. if a program executes in 30 seconds and 15 seconds of exec. uses enhancement, fraction = ½ (always < 1)**
  - **2nd…**
    - **Improvement gained by enhancement (i.e. how much faster does the program run overall)**
    - **i.e. if enhanced task takes 3.5 seconds and original task took 7, we say the speedup is 2 (always > 1)**

Deriving the previous formula

$$Speedup_{overall} = \frac{\text{Execution Time}_{old}}{\text{Execution Time}_{new}} = \frac{1}{(1 - Fraction_{enhanced}) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}}}$$

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

1 ← normalized old execution time

$(1 - Fraction_{enhanced})$: 1 - % enhanced (i.e. part of the task will take the same amount of time as before)

$\frac{Fraction_{enhanced}}{Speedup_{enhanced}}$ → $\frac{\text{\% of task that will run faster}}{\text{how much faster it will run}}$

(note: # should be < 1)
(otherwise, performance gets worse)
(represents new component of ex. time)

Amdahl’s Law examples