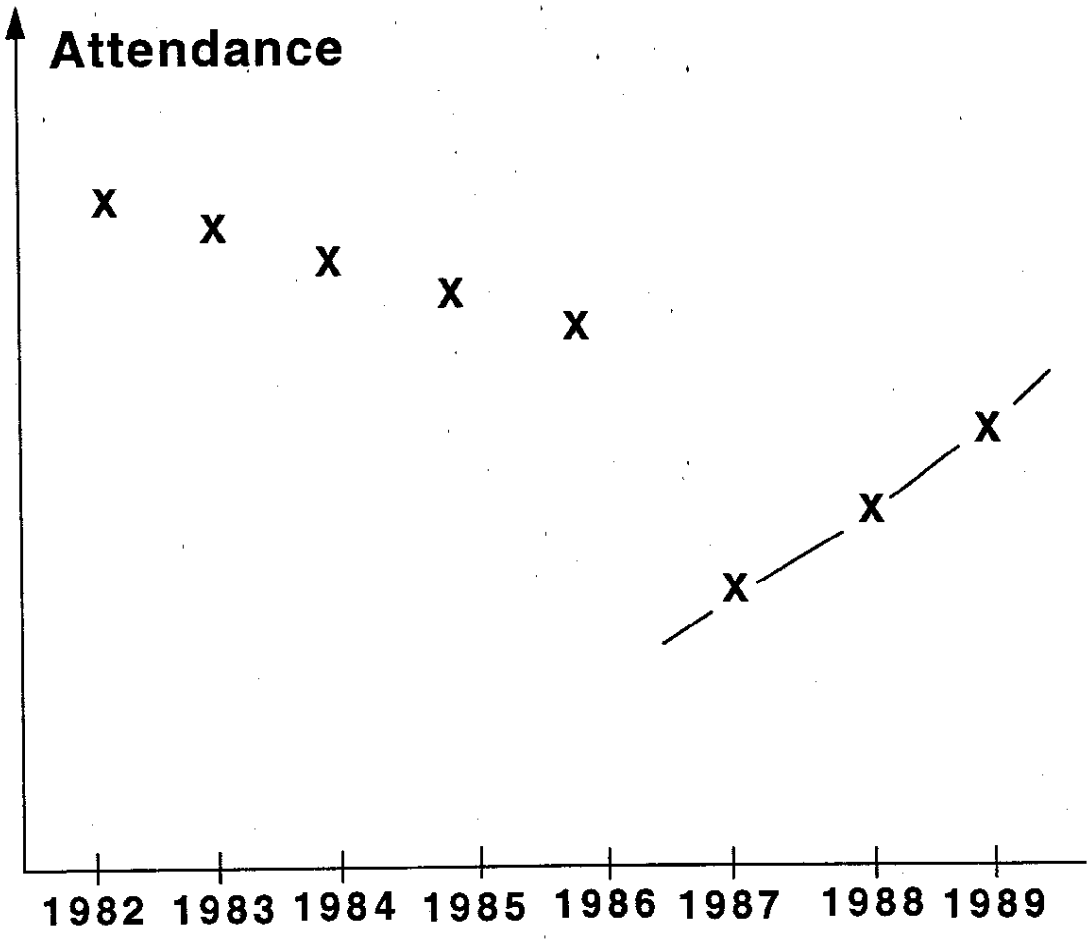
Introduction to

Measurement Methodology

Outline

* **Introduction**
  - **Misuse of the data**
  - **The Basic Equation (how long did it take)**
  - **The Mean**
* **How do we Measure**
  - **Real Hardware, Simulator, Analytical Model**
  - **Hardware Instrument, μcode, Software Monitor**
* **What do we Measure (Benchmarks)**
  - **Synthetic code**
  - **Kernels**
  - **Toy Benchmarks**
  - **SPEC**
  - **The Perfect Club**
  - **Your Relevant Workload**
* **Serious Abuses**

From a Welcoming Address At A Well-Known Conference

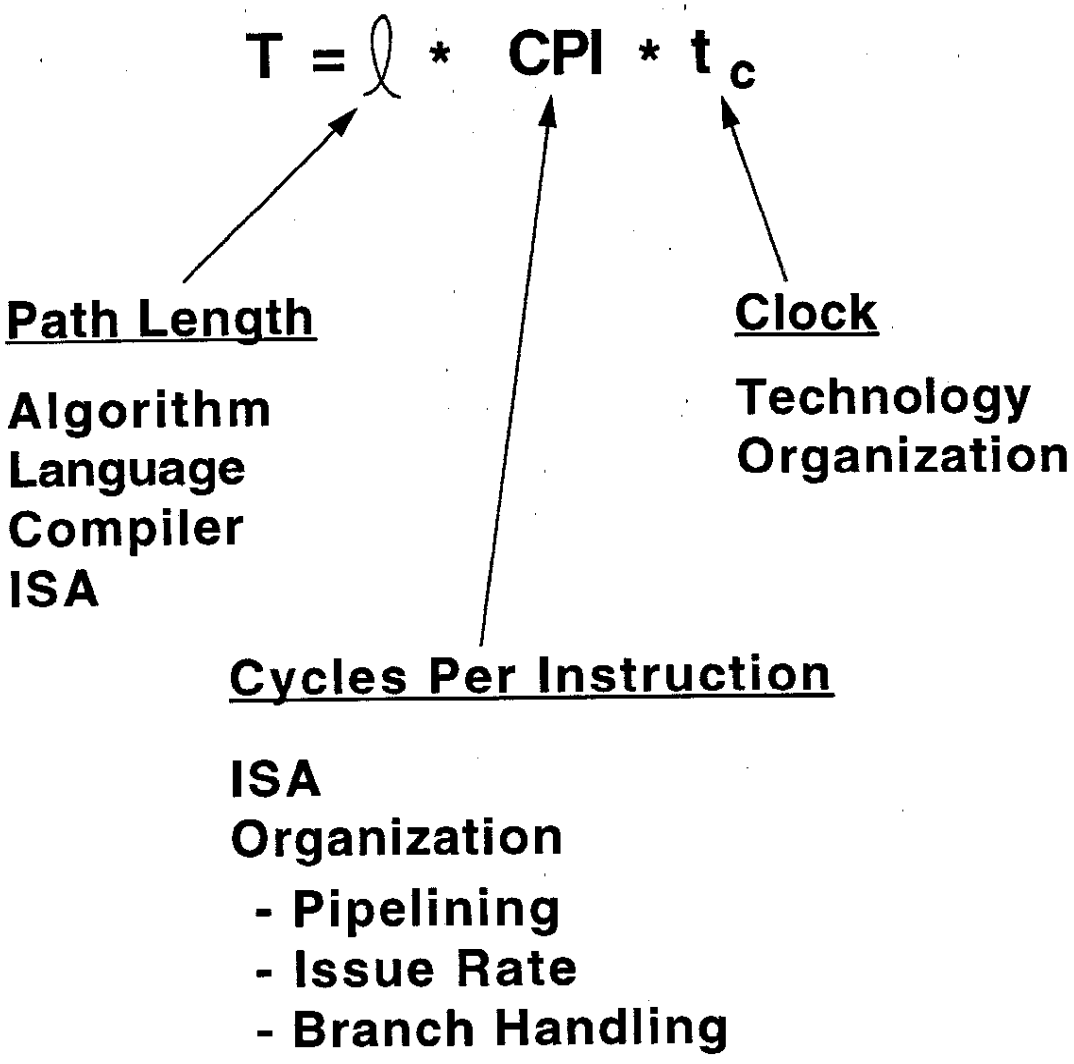


Why Measure

* **Before the fact**
  - **So we know what to build**

* **After the fact**
  - **So we know what to do next time**

The Standard Performance Equation



Means

* **Arithmetic Mean**

$$A = \frac{1}{n} \sum_{i=1}^{n} P_i$$

* **Geometric Mean**

$$G = \sqrt[n]{\prod_{i=1}^{n} P_i}$$

* **Harmonic Mean**

$$H = \frac{1}{\frac{1}{n} \sum_{i=1}^{n} \frac{1}{P_i}}$$

Why Harmonic Means Work for Rates

If we are dealing with performance, As measured in Megaflops,

M_i = Megaflops on Benchmark i

If all benchmarks are approximately equal with respect to amount of work,

$$M_i = \frac{F}{T_i}$$ (F is work per benchmark)

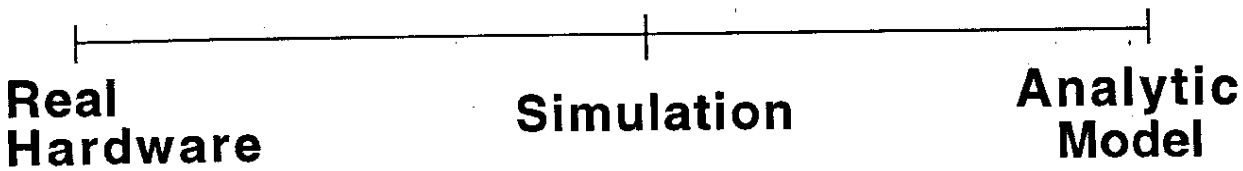
$$\text{Then, } H = \frac{1}{\frac{1}{n}\sum_{i=1}^{n}\frac{1}{M_i}} = \frac{1}{\frac{1}{nF}\sum_{i=1}^{n} T_i}$$

$$H = \frac{nF}{\sum_{i=1}^{n} T_i}$$

(Total Work divided by Total Time)

How Do We Measure

Degree of Santizing



Real Hardware

- "Gotchas" Have a chance to get in the way
- Least Flexible
- Fast for doing thorough job

Simulation

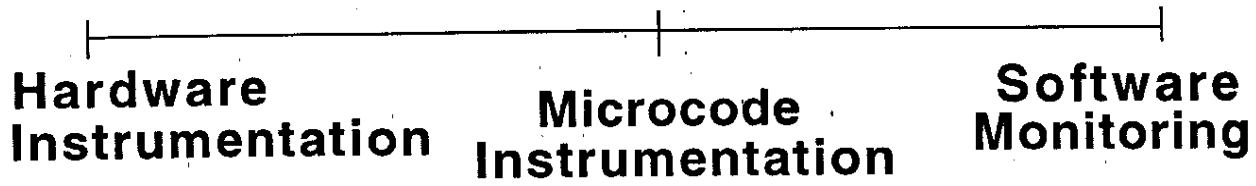
- Some effects are missing
- Most Flexible
- Slowest

Analytic Model

- Good for gross effects
- Must be validated

How Do We Measure (Continued)

Invasiveness



Hardware Instrumentation

- Most Expensive
- Non-Invasive
- Least Flexible

Microcoded Instrumentation

- Best of Both Worlds
- SPAM

Software Monitoring

- Cheap
- Very Invasive
- Most Flexible

Benchmarks

Rationale: Find a set of programs or program fragments representative of the workload you will be requiring of the machine

Types:

1. The ADD instruction - very old
2. Instruction MIX - Old (Gibson MIX, 1959)
3. Kernels
   - e.g., Livermore Loops
4. Synthetic Benchmarks
   - Parameterized
   - Careful: RRW is not RWR
5. Toy Benchmarks
   - Easy to hand-compile
   - Pretty much in disrepute today
     e.g., Towers of Hanoi
6. SPEC Suite (Systems Performance Evaluation Co-operative)
   - At least common agreement, I Guess!!
7. Real Workload

A few of my concerns

* ***One number: SpecMARK***
  -- Better than ADD time?

* ***SimplScalar***
  -- the entry bar
  -- the panel
  — bugs
* ***In the literature***
  -- 1.85 IPC max
  -- Issue width does not matter

* ***400 floating point ops or 1 L2 miss***

* ***Power models***

* ***IPC ...or CPI?***
  Does it matter?
  (Are you in Marketing, or
  Are you in Engineering?)

Bad Ways to Measure Performance (... and each has been used and reported in the Open Literature)

* **Apples & Oranges**
  - A Lightly Loaded VAX vs. Counting Simulated Cycles
* **Who Gets the Credit**
  - The Architecture or the Compiler
  - Example: Berkeley Pascal vs VMS Pascal
  - Algorithm Optimizations
  - Instruction set or register windows (Colwell)
* **Choice on Benchmarks**
  - Selective
    * Overstates significance of one feature
      e.g. Regularity (Fl. Pt.)
      e.g. Procedure Call Intensive
      e.g. No Floating Point
  - Small
    * 100% Cache, TB Hits
    * No I/O, Context Switch

★ Play with Statistics

	Program A	Program B
Machine 1:	1 unit	2 units
Machine 2:	2 units	1 unit

Machine 1 is $\frac{2}{1}$ on A, $\frac{1}{2}$ on B

Speed Up is $\frac{1}{2}$ $(2 + \frac{1}{2}) = 1.25$

★ Too Focused on Frequency

	Frequency	Execution Time
Calls	2.5%	21.6%
MOVL	12.4%	6.8%

The Dhrystone MIPS Joke

* **Dhrystone - A 300 Line Synthetic Benchmark**
  - Small main program, 11 short subroutines
  - "Typical" Frequencies of common ops.
    - Arithmetic
    - Loop Control
    - Subroutine Calls
  - No input data

* **Reference: VAX-11/780 on a 1985 Compiler achieved 1757 dhrystones**

* **The Metric:** $\frac{MIPS_i}{DHRY_i} = \frac{1}{1757}$

* **The Problem:**
  - Run Dhrystone with local optimir: 680 INST/ITER
  - Variables in Registers: 461 INST/ Iteration
  - Classical Optimizations (Global): 407
  - Inlining + full optimizations: 297
  - Theoretical limit (no input data): 0