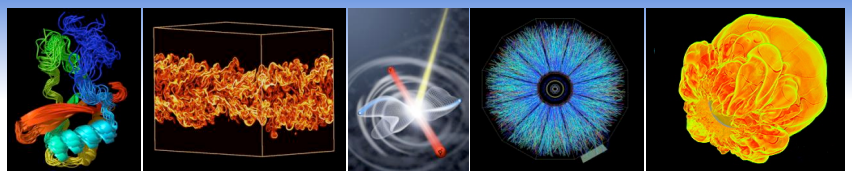


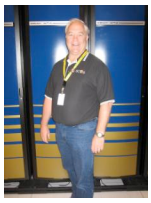
inst.eecs.berkeley.edu/~cs61c

# UCB CS61C : Machine Structures

## Lecture 28 – Performance and Benchmarks 2008-08-11

Bill Kramer

August 11, 2008



## Why Measure Performance? Faster is better!

- **Purchasing Perspective: given a collection of machines (or upgrade options), which has the**
  - **best performance ?**
  - **least cost ?**
  - **best performance / cost ?**
- **Computer Designer Perspective: faced with design options, which has the**
  - **best performance improvement ?**
  - **least cost ?**
  - **best performance / cost ?**
- **All require a basis for comparison and metric(s) for evaluation!**
  - **Solid metrics lead to solid progress!**


## Notions of "Performance"

| Plane | DC to Paris | Top Speed | Passen-gers | Throughput (pmph) |
|---|---|---|---|---|
| Boeing 747 | 6.5 hours | 610 mph | 470 | 286,700 |
| BAD/Sud Concorde | 3 hours | 1350 mph | 132 | 178,200 |

- Which has higher performance? What is the performance
  - Interested in time to deliver 100 passengers?
  - Interested in delivering as many passengers per day as possible?
  - Which has the best price performance? (per $, per Gallon)
- In a computer, time for one task called
  Response Time or Execution Time
- In a computer, tasks per unit time called
  Throughput or Bandwidth


## Definitions

- Performance is in units of things per time period
  - higher is better
- If mostly concerned with response time

$$Performance(X) = \frac{1}{execution\_time(X)}$$

- " F(ast) is *n* times faster than S(low) " means:

$$n = \frac{Performance(F)}{Performance(S)} = \frac{Execution\_time(S)}{Execution\_time(F)}$$

## Example of Response Time v. Throughput

- Time of Concorde vs. Boeing 747?
  - Concord is 6.5 hours / 3 hours = 2.2 times faster
  - Concord is 2.2 times ("120%") faster in terms of flying time (response time)
- Throughput of Boeing vs. Concorde?
  - Boeing 747: 286,700 passenger-mph / 178,200 passenger-mph = 1.6 times faster
  - Boeing is 1.6 times ("60%") faster in terms of throughput
- We will focus primarily on response time.

## Words, Words, Words…

- Will (try to) stick to "n times faster"; its less confusing than "m % faster"
- As faster means both decreased execution time and increased performance,
  - to reduce confusion we will (and you should) use "improve execution time" or "improve performance"

## What is Time?

- Straightforward definition of time:
  - Total time to complete a task, including disk accesses, memory accesses, I/O activities, operating system overhead, ...
  - "real time", "response time" or "elapsed time"
- Alternative: just the time the processor (CPU) is working only on your program (since multiple processes running at same time)
  - "CPU execution time" or "CPU time"
  - Often divided into system CPU time (in OS) and user CPU time (in user program)

## How to Measure Time?

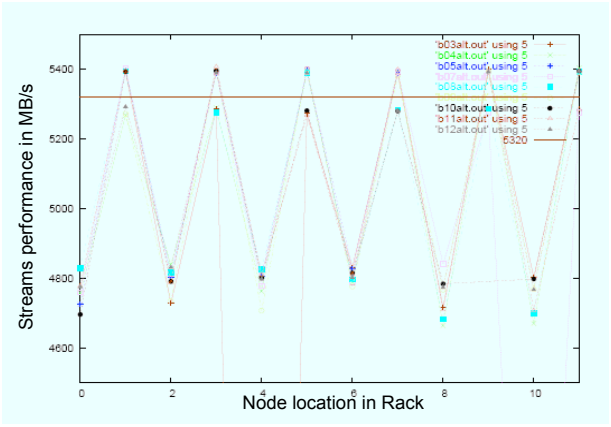
- Real Time ⇒ Actual time elapsed

- CPU Time: Computers constructed using a clock that runs at a constant rate and determines when events take place in the hardware
  - These discrete time intervals called clock cycles (or informally clocks or cycles)
  - Length of clock period: clock cycle time (e.g., ½ nanoseconds or ½ ns) and clock rate (e.g., 2 gigahertz, or 2 GHz), which is the inverse of the clock period; use these!

## Measuring Time Using Clock Cycles (1/2)

- **CPU execution time for a program**
  - **Units of [seconds / program] or [s/p] = Clock Cycles for a Program x Clock Period**
  - **Units of [s/p] = [cycles / p] x [s / cycle] = [c/p] x [s/c]**
- **Or**
  **= Clock Cycles for a program [c / p]**
  **~~Clock Rate~~ [c / s]**

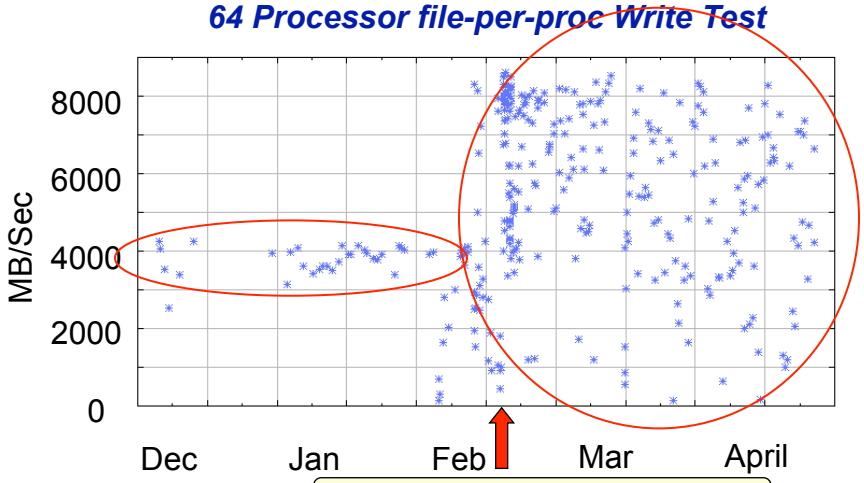

## Real Example of Why Testing is Needed Hardware configuration choices




**Memory test performance depends where the adaptor is plugged in.**

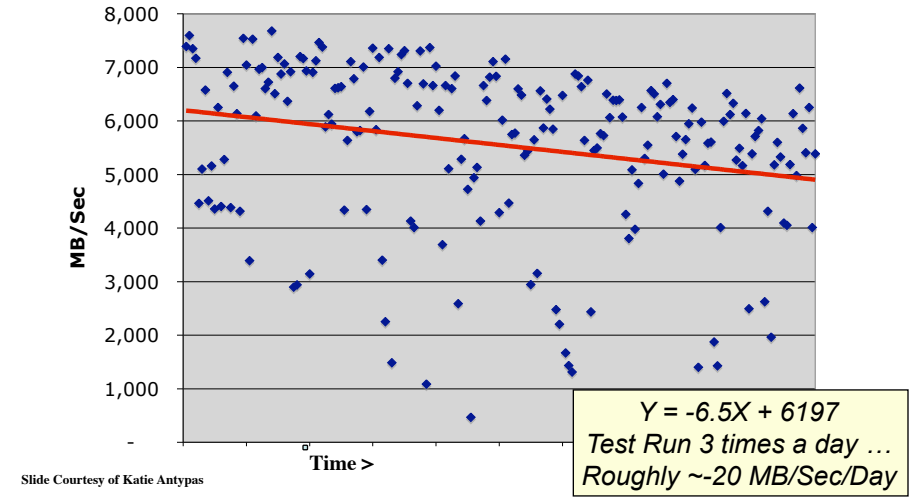
# Real Example - IO Write Performance

2X performance improvement - 4X consistency decrease




Slide Courtesy of Katie Antypas

# Real Example Read Performance Degrades Over Time




Slide Courtesy of Katie Antypas

## Measuring Time using Clock Cycles (2/2),

- One way to define clock cycles:
  - Clock Cycles for program [c/p]
    = Instructions for a program [i/p]
    (called "Instruction Count")
    x Average Clock cycles Per Instruction [c/i]
    (abbreviated "CPI")
- CPI one way to compare two machines with same instruction set, since Instruction Count would be the same

## Performance Calculation (1/2)

- CPU execution time for program [s/p]
  = Clock Cycles for program [c/p]
  x Clock Cycle Time [s/c]

- Substituting for clock cycles:
  CPU execution time for program [s/p]
  = ( Instruction Count [i/p] x CPI [c/i] )
  x Clock Cycle Time [s/c]

  = Instruction Count x CPI x Clock Cycle Time


## Performance Calculation (2/2)

$$CPUTime = \frac{Instuctions}{Program} \times \frac{Cycles}{Instruction} \times \frac{Seconds}{Cycle}$$

$$CPUTime = \frac{\cancel{Instuctions}}{Program} \times \frac{\cancel{Cycles}}{\cancel{Instruction}} \times \frac{Seconds}{\cancel{Cycle}}$$

$$CPUTime = \frac{Seconds}{Program}$$

**Product of all 3 terms: if missing a term, cannot predict the time, the real measure of performance**


## How Calculate the 3 Components?

- **Clock Cycle Time: in specification of computer**
  - **(Clock Rate in advertisements)**
- **Instruction Count:**
  - **Count instructions in loop of small program**
  - **Use simulator to count instructions**
  - **Hardware performance counters in special register**
    - **(Pentium II,III,4, etc.)**
    - **Performance API (PAPI)**
- **CPI:**
  - **Calculate: Execution Time / Clock cycle time**
    **Instruction Count**
  - **Hardware counter in special register (PII,III,4, etc.)**


## Calculating CPI Another Way

- First calculate CPI for each individual instruction (`add`, `sub`, `and`, etc.)
- Next calculate frequency of each individual instruction
- Finally multiply these two for each instruction and add them up to get final CPI (the weighted sum)


## Example (RISC processor)

| Op | $Freq_i$ | $CPI_i$ | Prod | (% Time) |
|---|---|---|---|---|
| ALU | 50% | 1 | .5 | (23%) |
| Load | 20% | 5 | 1.0 | (45%) |
| Store | 10% | 3 | .3 | (14%) |
| Branch | 20% | 2 | .4 | (18%) |
| | | | **2.2** | |

Instruction Mix (Where time spent)

- **What if Branch instructions twice as fast?**

## Answer (RISC processor)

| Op | $Freq_i$ | $CPI_i$ | Prod | (% Time) |
|---|---|---|---|---|
| ALU | 50% | 1 | .5 | (25%) |
| Load | 20% | 5 | 1.0 | (50%) |
| Store | 10% | 3 | .3 | (15%) |
| Branch | 20% | 1 | .2 | (10%) |
| | | | **2.0** | |

Instruction Mix

(Where time spent)

## Administrivia

- Tuesday's lab
  - 11-1,3-5,5-7 meeting as normal
    - Get lab 14 checked off
    - Can ask final review questions to TA
- Review session
  - Tuesday 1-3pm, location TBD, check website
- Proj3 grading
  - No face to face, except special cases

# Issues of Performance Understand

*Current Methods of Evaluating HPC systems are incomplete and may be insufficient for the future highly parallel systems.*

- Because
  - Parallel Systems are complex, multi-faceted systems
    - Single measures can not address current and future complexity
  - Parallel systems typically have multiple application targets
    - Communities and applications getting broader
  - Parallel requirements are more tightly coupled than many systems
  - Point evaluations do not address the life cycle of a living system
    - On-going usage
    - On-going system management
    - HPC Systems are not stagnant
      - Software changes
      - New components - additional capability or repair
      - Workload changes

# The PERCU Method
## *What Users Want*

- Performance
  - How fast will a system process work if everything is working really well
- Effectiveness
  - The likelihood users can get the system to do their work when they need it
- Reliability
  - The likelihood the system is available to do the user's work
- Consistency
  - How often the system processes the same or similar work correctly and in the same length of time
- Usability
  - How easy is it for users to get the system to process their work as fast as possible

PERCU


## The Use of Benchmarks

- A Benchmark is an application and a problem that jointly define a test.
- Benchmarks should efficiently serve four purposes
  - Differentiation of a system from among its competitors
    - System and Architecture studies
    - Purchase/*selection*
  - Validate that a system works the way expected once a system is built and/or is delivered
  - Assure that systems perform as expected throughout its lifetime
    - e.g. after upgrades, changes, and in regular use
  - Guidance to future system designs and implementation


## What Programs Measure for Comparison?

- Ideally run typical programs with typical input before purchase, or before even build machine
  - Called a "workload";
  - For example:
  - Engineer uses compiler, spreadsheet
  - Author uses word processor, drawing program, compression software
- In some situations are hard to do
  - Don't have access to machine to "benchmark" before purchase
  - Don't know workload in future

# Benchmarks

- Apparent sustained speed of processor depends on code used to test it
- Need industry standards so that different processors can be fairly compared
  - Most "standard suites" are simplified
    - Type of algorithm
    - Size of problem
    - Run time
- Organizations exist that create "typical" code used to evaluate systems
- Tests need changed every ~5 years (HW design cycle time) since designers could (and do!) target specific HW for these standard benchmarks
  - This HW may have little or no general benefit

# Choosing Benchmarks

**Examine Application Workload**

- Understand user requirements
- Science Areas
- Algorithm Spaces allocation goals
- Most run codes

- Benchmarks often have to be simplified
  - Time and resources available
  - Target systems

**Find Representative Applications**

- Good coverage in science areas, algorithm space, libraries and language
- Local knowledge helpful
- Freely available
- Portable, challenging, but not impossible for vendors to run

- Benchmarks must
  - Look to the past
    - Past workload and Methods
  - Look to the future
    - Future Algorithms and Applications
    - Future Workload Balance

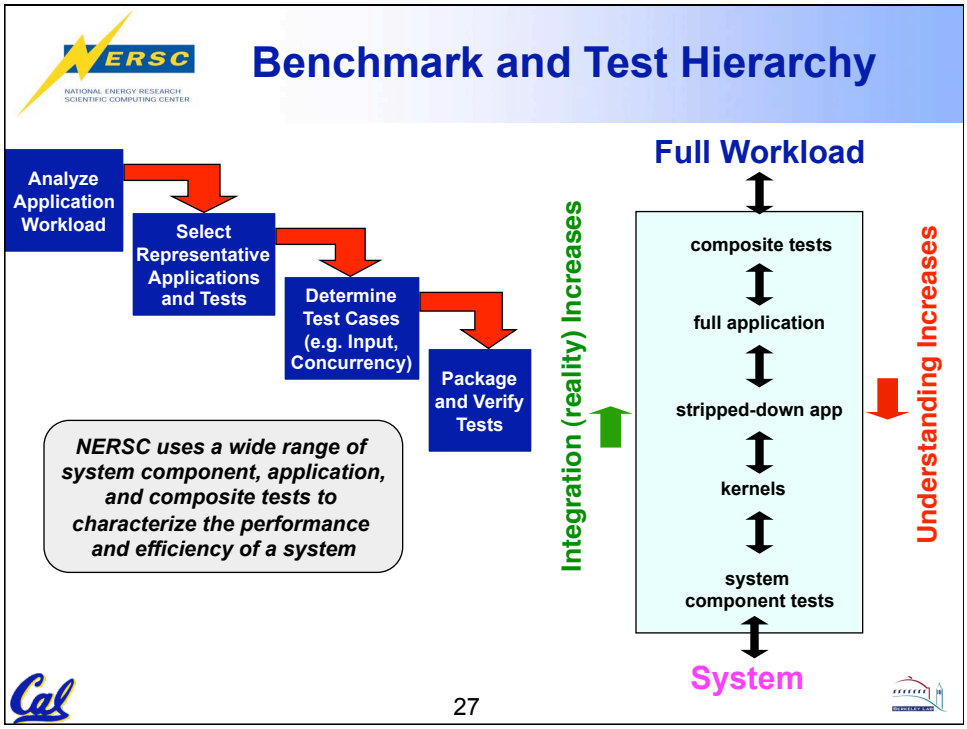
**Determine Concurrency & Inputs**

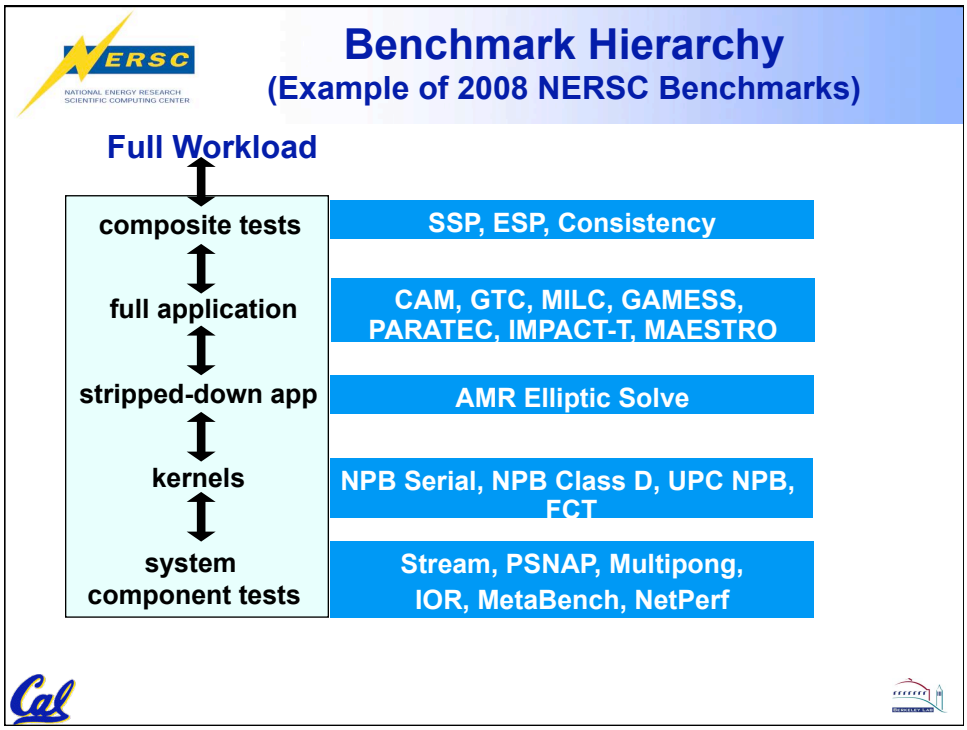
- Aim for upper end of application's concurrency limit today
- Determine correct problem size and inputs
- Balance desire for high concurrency runs with likelihood of getting real results rather than projections
- Create weak or strong scaling problems

**Test, Benchmark and Package**

- Test chosen benchmarks on multiple platforms
- Characterize performance results
- Create verification test
- Prepare benchmarking instructions and package code for vendors

- Workload Characterization Analysis (WCA) - a statistical study of the applications in a workload
  - More formal and Lots of Work
  - Workload Analysis with Weights (WAW) - after a full WCA
- Sample Estimation of Relative Performance of Programs (SERPOP)
  - Common - particularly with Suites of standard BMs
  - Most often Not weighted


Benchmark and Test Hierarchy
Analyze Application Workload
Select Representative Applications and Tests
Determine Test Cases (e.g. Input, Concurrency)
Package and Verify Tests
NERSC uses a wide range of system component, application, and composite tests to characterize the performance and efficiency of a system
Full Workload
composite tests
full application
stripped-down app
kernels
system component tests
System
Integration (reality) Increases
Understanding Increases
27


Benchmark Hierarchy
(Example of 2008 NERSC Benchmarks)
Full Workload
composite tests
full application
stripped-down app
kernels
system component tests
SSP, ESP, Consistency
CAM, GTC, MILC, GAMESS, PARATEC, IMPACT-T, MAESTRO
AMR Elliptic Solve
NPB Serial, NPB Class D, UPC NPB, FCT
Stream, PSNAP, Multipong, IOR, MetaBench, NetPerf

## Example Standardized Benchmarks (1/2)

- Standard Performance Evaluation Corporation (SPEC) SPEC CPU2006
  - CINT2006 12 integer (perl, bzip, gcc, go, ...)
  - CFP2006 17 floating-point (povray, bwaves, ...)
  - All relative to base machine (which gets 100)
    e.g Sun Ultra Enterprise 2 w/296 MHz UltraSPARC II
  - They measure
    - System speed (SPECint2006)
    - System throughput (SPECint_rate2006)
  - www.spec.org/osg/cpu2006/

## Example Standardized Benchmarks (2/2)

- SPEC
  - Benchmarks distributed in source code
  - Members of consortium select workload
    - 30+ companies, 40+ universities, research labs
  - Compiler, machine designers target benchmarks, so try to change every 5 years
  - SPEC CPU2006:

CINT2006

| | | |
|---|---|---|
| perlbench | C | Perl Programming language |
| bzip2 | C | Compression |
| gcc | C | C Programming Language Compiler |
| mcf | C | Combinatorial Optimization |
| gobmk | C | Artificial Intelligence : Go |
| hmmer | C | Search Gene Sequence |
| sjeng | C | Artificial Intelligence : Chess |
| libquantum | C | Simulates quantum computer |
| h264ref | C | H.264 Video compression |
| omnetpp | C++ | Discrete Event Simulation |
| astar | C++ | Path-finding Algorithms |
| xalancbmk | C++ | XML Processing |

CFP2006

| | | |
|---|---|---|
| bwaves | Fortran | Fluid Dynamics |
| gamess | Fortran | Quantum Chemistry |
| milc | C | Physics / Quantum Chromodynamics |
| zeusmp | Fortran | Physics / CFD |
| gromacs | C,Fortran | Biochemistry / Molecular Dynamics |
| cactusADM | C,Fortran | Physics / General Relativity |
| leslie3d | Fortran | Fluid Dynamics |
| namd | C++ | Biology / Molecular Dynamics |
| dealII | C++ | Finite Element Analysis |
| soplex | C++ | Linear Programming, Optimization |
| povray | C++ | Image Ray-tracing |
| calculix | C,Fortran | Structural Mechanics |
| GemsFDTD | Fortran | Computational Electromegnetics |
| tonto | Fortran | Quantum Chemistry |
| lbm | C | Fluid Dynamics |
| wrf | C,Fortran | Weather |
| sphinx3 | C | Speech recognition |

## Another Benchmark Suite

- **NAS Parallel Benchmarks** (http://www.nas.nasa.gov/News/Techreports/1994/PDF/RNR-94-007.pdf)
  - 8 parallel codes that represent "psuedo applications" and kernels
    - Multi-Grid (MG)
    - Conjugate Gradient (CG)
    - 3-D FFT PDE (FT)
    - Integer Sort (IS)
    - LU Solver (LU)
    - Pentadiagonal solver (SP)
    - Block Tridiagional Solver (BT)
    - Embarrassingly Parallel (EP)
  - Originated as "pen and paper" tests (1991) as early parallel systems evolved
    - Defined a problem and algorithm
    - Now there are reference implementations (Fortran, C)/(MPI, OPenMP, Grid, UPC)
    - Can set any concurrency

## Other Benchmark Suites

- **TPC -** Transaction Processing
- **IOR:** Measure I/O throughput
  - Parameters set to match sample user applications
  - Validated performance predictions in SC08 paper
- **MetaBench:** Measures filesystem metadata transaction performance
- **NetPerf:** Measures network performance
- **Stream:** Measures raw memory bandwidth
- **PSNAP(TPQX):** Measures idle OS noise and jitter
- **Multipong:** Measure interconnect latency and bandwidth from nearest to furthest node
- **FCT - Full-configuration test**
  - 3-D FFT - Tests ability to run across entire system of any scale
- **Net100 - Network implementations**
- **Web100 - Web site functions**

# Algorithm Diversity

| Science areas \ Algorithm | Dense linear algebra | Sparse linear algebra | Spectral Methods (FFT)s | Particle Methods | Structured Grids | Unstructured or AMR Grids | Data Intensive |
|---|---|---|---|---|---|---|---|
| Accelerator Science | | X | X | X | X | X | |
| Astrophysics | X | X | | X | X | X | |
| Chemistry | | X | | X | | X | |
| Climate | | X | | X | X | X | |
| Combustion | | | | | X | X | |
| Fusion | | X | | X | X | X | |
| Lattice Gauge | | X | | X | X | | |
| Material Science | X | | X | X | X | | |

High Flop/s rate

High performance memory system

High bisection bandwidth

High performance memory system

High flop/s rate

Low latency, efficient gather/ scatter

Storage, Network Infrastructure

***Many users require a system which performs adequately in all areas***

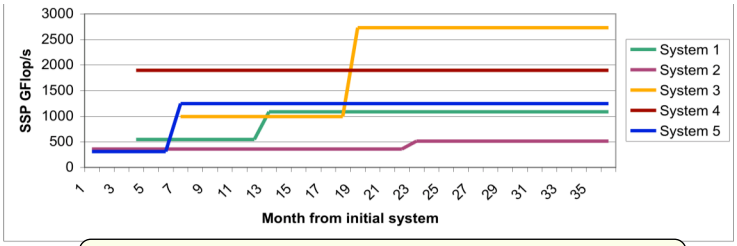
33

# Sustained System Performance

- The ***"If I wait the technology will get better"*** syndrome
- Measures mean flop rate of applications integrated over time period
- SSP can change due to
  - System upgrades, Increasing # of cores, Software Improvements
- Allows evaluation of systems delivered in phases
- Takes into account delivery date
- Produces metrics such as SSP/Watt and SSP/$

$$Value_s = \frac{Potency_s}{Cost_s}$$

*SSP Over 3 Year Period for 5 Hypothetical Systems*




*Area under curve, when combined with cost, indicates system 'value'*

# Example of spanning Application Characteristics

| Benchmark | Science Area | Algorithm Space | Base Case Concurrency | Problem Description | Memory | Lang | Libraries |
|---|---|---|---|---|---|---|---|
| CAM | Climate (BER) | Navier Stokes CFD | 56, 240 Strong scaling | D Grid, (~.5 deg resolution); 240 timesteps | 0.5 GB *per MPI task* | F90 | netCDF |
| GAMESS | Quantum Chem (BES) | Dense linear algebra | 256, 1024 (Same as TI-09) | DFT gradient, MP2 gradient | ~2GB *per MPI task* | F77 | DDI, BLAS |
| GTC | Fusion (FES) | PIC, finite difference | 512, 2048 Weak scaling | 100 particles per cell | .5 GB*per MPI task* | F90 | |
| IMPACT-T | Accelerator Physics (HEP) | PIC, FFT component | 256,1024 Strong scaling | 50 particles per cell | 1 GB *per MPI task* | F90 | |
| MAESTRO | Astrophysics (HEP) | Low Mach Hydro; block structured-grid multiphysics | 512, 2048 Weak scaling | 16 32^3 boxes per proc; 10 timesteps | 800-1GB *per MPI task* | F90 | Boxlib |
| MILC | Lattice Gauge Physics (NP) | Conjugate gradient, sparse matrix; FFT | 256, 1024, 8192 Weak scaling | 8x8x8x9 Local Grid, ~70,000 iterations | 210 MB *per MPI task* | C, assem. | |
| PARATEC | Material Science (BES) | DFT; FFT, BLAS3 | 256, 1024 Strong scaling | 686 Atoms, 1372 bands, 20 iterations | .5 -1GB *per MPI task* | F90 | Scalapack, FFTW |

# Time to Solution is the Real Measure

| Hypothetical N6 System | Tasks | System Gflopcnt | Results: Time | Results: Rate per Core |
|---|---|---|---|---|
| CAM | **240** | **57,669** | **408** | 0.589 |
| GAMESS | **1024** | **1,655,871** | **2811** | 0.575 |
| GTC | **2048** | **3,639,479** | **1493** | 1.190 |
| IMPACT-T | **1024** | **416,200** | **652** | 0.623 |
| MAESTRO | **2048** | **1,122,394** | **2570** | 0.213 |
| MILC | **8192** | **7,337,756** | **1269** | 0.706 |
| PARATEC | **1024** | **1,206,376** | **540** | 2.182 |
| GEOMETRIC MEAN | | | | 0.7 |

Rate Per Core = Ref. Gflop count / (Tasks*Time)

Flop count measured on reference system

Measured wall clock time on hypothetical system

Geometric mean of 'Rates per Core'

**SSP (TF) = Geometric mean of rates per core * # cores in system/1000**

## NERSC-6 Benchmarks Coverage

| Science areas | Dense linear algebra | Sparse linear algebra | Spectral Methods (FFT)s | Particle Methods | Structured Grids | Unstructured or AMR Grids |
|---|---|---|---|---|---|---|
| Accelerator Science | | | IMPACT-T | IMPACT-T | IMPACT-T | |
| Astrophysics | | MAESTRO | | | MAESTRO | MAESTRO |
| Chemistry | GAMESS | | | | | |
| Climate | | | CAM | | CAM | |
| Combustion | | | | | MAESTRO | AMR Elliptic |
| Fusion | | | | GTC | GTC | |
| Lattice Gauge | | MILC | MILC | MILC | MILC | |
| Material Science | PARATEC | | PARATEC | | PARATEC | |

37

## Performance Evaluation: An Aside Demo

**If we're talking about performance, let's discuss the ways shady salespeople have fooled consumers (so you don't get taken!)**

5. Never let the user touch it
4. Only run the demo through a script
3. Run it on a stock machine in which "no expense was spared"
2. Preprocess all available data
1. Play a movie

## David Bailey's "12 Ways to Fool the Masses"

1. Quote only 32-bit performance results, not 64-bit results.
2. Present performance figures for an inner kernel, and then represent these figures as the performance of the entire application.
3. Quietly employ assembly code and other low-level language constructs.
4. Scale up the problem size with the number of processors, but omit any mention of this fact.
5. Quote performance results projected to a full system (based on simple serial cases).
6. Compare your results against scalar, unoptimized code on Crays.
7. When direct run time comparisons are required, compare with an old code on an obsolete system.
8. If MFLOPS rates must be quoted, base the operation count on the parallel implementation, not on the best sequential implementation.
9. Quote performance in terms of processor utilization, parallel speedups or MFLOPS per dollar.
10. Mutilate the algorithm used in the parallel implementation to match the architecture.
11. Measure parallel run times on a dedicated system, but measure conventional run times in a busy environment.
12. If all else fails, show pretty pictures and animated videos, and don't talk about performance.

## Peak Performance Has Nothing to Do with Real Performance

| System | Cray XT-4 Dual Core | Cray XT-4 Quad Core | IBM BG/P | IBM Power 5 |
|---|---|---|---|---|
| Processor | AMD | AMD | Power PC | Power 5 |
| Peak Speed per processor | 5.2 Gflops/s = 2.6 GHz * 2 Instruction/ Clock | 9.2 Gflops/s = 2.3 GHz * 4 Instruction/Clock | 3.4 Gflops/s = .85 GHz * 4 Instructions/ Clock | 7.6 Gflops/s = 1.9 GHz * 4 Instructions/ Clock |
| Sustained per processor speed for NERSC SSP | 0.70 Gflops/s | 0.63 Gflops/s | 0.13 Gflops/s | 0.65 Gflops/ s |
| NERSC SSP % of peak | 13.4% | 6.9% | 4% | 8.5% |
| Approximate Relative Cost per Core | 2.0 | 1.25 | .5 | 6.1 |
| Year | 2006 | 2007 | 2008 | 2005 |

# Peer Instruction

A. Rarely does a company selling a product give unbiased performance data.

B. The Sieve of Eratosthenes and Quicksort were early effective benchmarks.

C. A program runs in 100 sec. on a machine, `mult` accounts for 80 sec. of that. If we want to make the program run 6 times faster, we need to up the speed of `mult`s by AT LEAST 6.

| | ABC |
|---|---|
| 0: | FFF |
| 1: | FFT |
| 2: | FTF |
| 3: | FTT |
| 4: | TFF |
| 5: | TFT |
| 6: | TTF |
| 7: | TTT |

# Peer Instruction Answers

TRUE

FALSE

FALSE

A. Rarely does a company selling a product give unbiased performance data.

B. The Sieve of Eratosthenes, Puzzle and Quicksort were early effective benchmarks.

C. A program runs in 100 sec. on a machine, mult accounts for 80 sec. of that. If we want to make the program run 6 times faster, we need to up the speed of mults by AT LEAST 6.

**A. TRUE. It is rare to find a company that gives Metrics that do not favor its product.**

**B. Early benchmarks? Yes. Effective? No. Too simple!**

**C. 6 times faster = 16 sec. mults must take -4 sec! I.e., impossible!**

| | ABC |
|---|---|
| 0: | FFF |
| 1: | FFT |
| 2: | FTF |
| 3: | FTT |
| 4: | TFF |
| 5: | TFT |
| 6: | TTF |
| 7: | TTT |

## "And in conclusion…"

$$CPUTime = \frac{Instuctions}{Program} \times \frac{Cycles}{Instruction} \times \frac{Seconds}{Cycle}$$

- **Latency vs. Throughput**
- **Performance doesn't depend on any single factor:** need Instruction Count, Clocks Per Instruction (CPI) and Clock Rate to get valid estimations
- **User Time: time user waits for program to execute: depends heavily on how OS switches between tasks**
- **CPU Time: time spent executing a single program: depends solely on design of processor (datapath, pipelining effectiveness, caches, etc.)**
- **Benchmarks**
  - **Attempt to understand (and project) performance,**
  - **Updated every few years**
  - **Measure everything from simulation of desktop graphics programs to battery life**
- **Megahertz Myth**
  - **MHz ≠ performance, it's just one factor**

## Megahertz Myth Marketing Movie

**http://www.youtube.com/watch?v=PKF9GOE2q38**