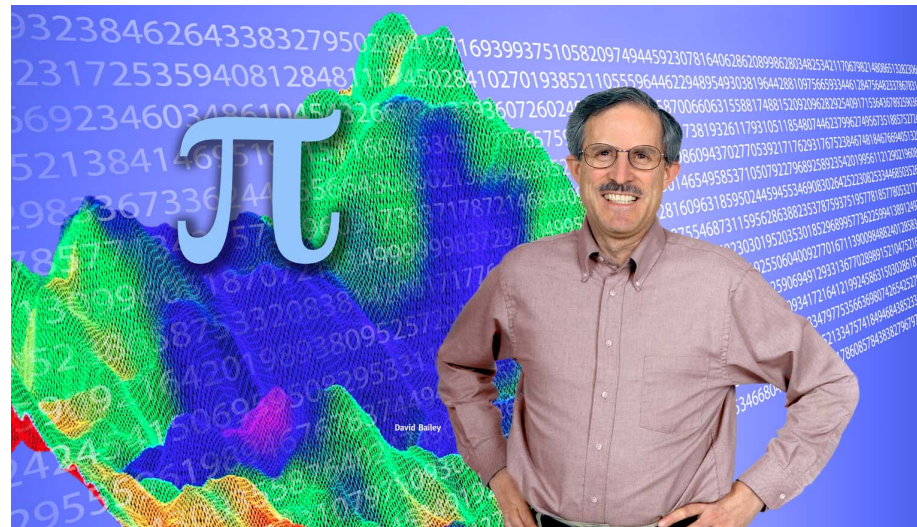

Twelve ways to fool the masses: Fast forward to 2013

David H Bailey

Lawrence Berkeley National Laboratory, USA (recently retired)
University of California, Davis, Department of Computer Science

This talk is available at:

<http://www.davidhbailey.com/dhbtalks/dhb-12ways-2013.pdf>



The need for reproducibility in scientific computing

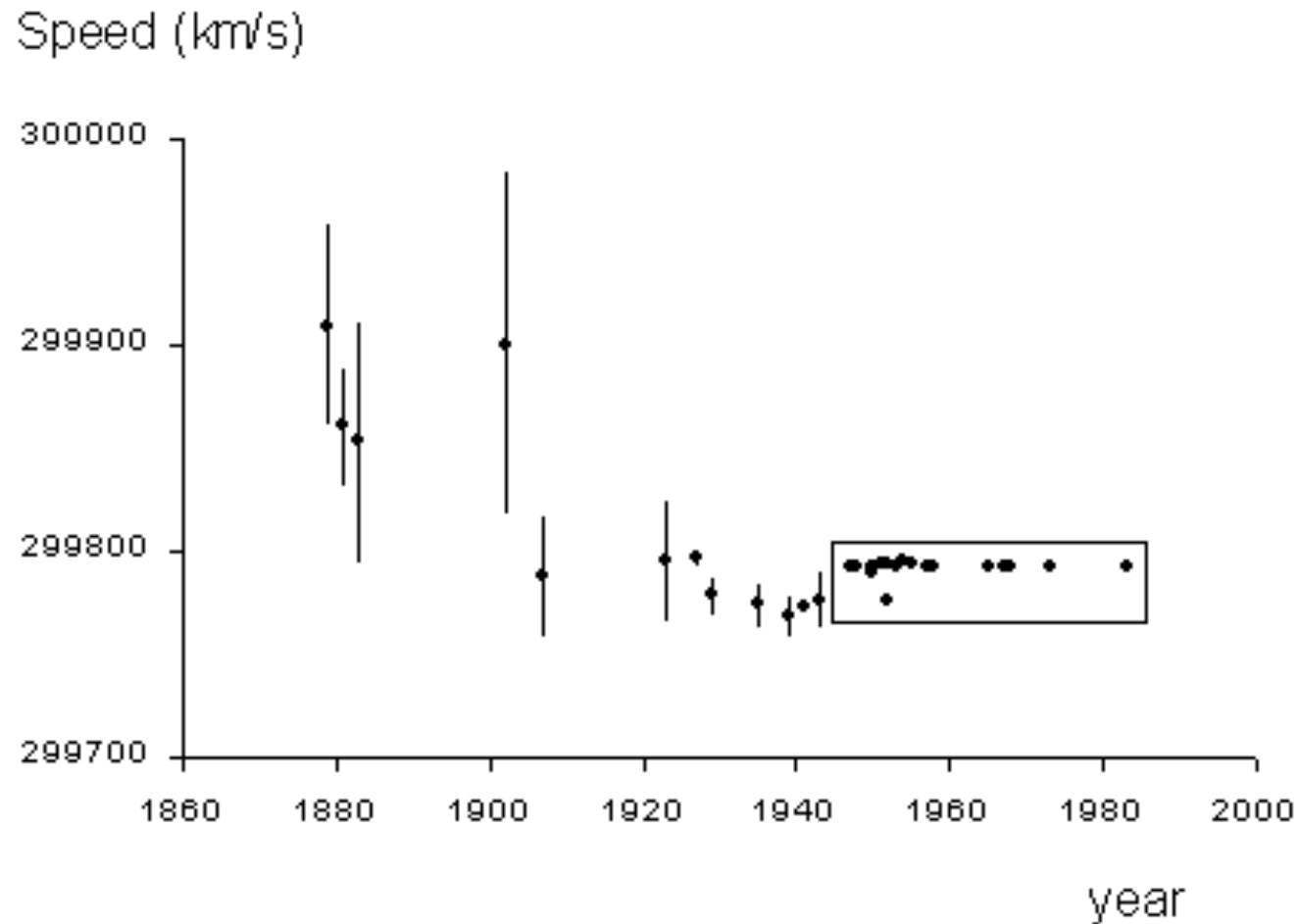
Computing is now widely recognized as the third leg of science, with theory and experiment. But the scientific computing culture has not kept pace:

- ◆ The notion that a computation should be reproducible by other researchers is still not part of the research culture.
- ◆ Physicists, chemists and biologists routinely keep notebooks of every detail, but few computational experiments are performed with such care.
- ◆ Scientific computing has its own concerns:
 - Is numeric precision adequate? Are algorithms appropriate?
 - Are there difficulties with numerical reproducibility? How many digits?
 - What independent validity checks have been employed?
 - Has the experiment been performed with different hardware/software?
 - Can results be reproduced with code written by other researchers?

For further information, see:

V. Stodden, D.H. Bailey, J.M. Borwein, R.J. LeVeque, W. Rider and W. Stein, "Setting the default to reproducible: Reproducibility in computational and experimental mathematics," 2 Feb 2013, available at: <http://www.davidhbailey.com/dhbpapers/icerm-report.pdf>

Example from physics: Measurements of speed of light



Why is there such a discrepancy between pre-1945 and post-1945 values?
This is probably due to biases and sloppy experimental methods.

Example from psychology: The “blank slate” paradigm

The “blank slate” paradigm (1920-1980):

- ◆ The human mind at birth is a “blank slate.”
- ◆ Heredity and biology play no significant role in human psychology – all personality and behavioral traits are socially constructed.

Current consensus, based on latest research:

- ◆ Humans at birth possess sophisticated facilities for social interaction, language acquisition, pattern recognition, navigation and counting.
- ◆ Heredity, evolution and biology are major factors in human personality.
- ◆ Some personality traits are more than 50% heritable.

How did the early 20th century scientists get it so wrong?

- ◆ Sloppy experimental methodology and analysis.
- ◆ Pervasive wishful thinking and “politically correct” biases.

Ref: Steven Pinker, [The Blank Slate: The Modern Denial of Human Nature](#)

Example from anthropology: The “noble savage” paradigm

Anthropologists, beginning with Margaret Mead in the 1930s, painted an idyllic picture of primitive societies (e.g., Pacific Islanders):

- ◆ Virtually no violence, jealousy or warfare.
- ◆ Happy, uninhibited – few of the psychological or social problems that afflict much of Western society.

But beginning in the 1970s, a new generation of anthropologists revisited these societies and did more careful studies. They found:

- ◆ Crime rates higher than most U.S. and European cities.
- ◆ Death rates from inter-tribe conflicts typically that exceed those of warfare among Western nations by factors of 10 or more.
- ◆ Psychological problems comparable to Western societies.
- ◆ Complex, jealous taboos surrounding courtship and marriage:
 - Violent reprisals in cases of adultery or non-virgin brides.
 - Ornaments worn by males of one tribe, earlier thought to be signs of male-female role reversal, were actually badges of honor from warfare.

Why so many errors? “Anthropological malpractice” – Pinker.

Lessons from history

- ◆ Research must be based on solid empirical tests and careful, objective analysis of data.
- ◆ Researchers must provide all details of the experimental environment, so others can reproduce their results.
- ◆ Rigorous peer review is essential.
- ◆ Good intentions or “politically correct” conclusions are no excuses for poor scholarship.
- ◆ Erudite technical terminology and mathematical formulas are no substitutes for sound reasoning.
- ◆ Hype has no place in the scientific enterprise.
- ◆ High standards of honesty and rigor must be vigilantly enforced.

Unless great care is taken, we can fool ourselves as well as others!

History of parallel computing

- ◆ 1976-1986: Initial research studies and demonstrations.
- ◆ 1986-1990: First large-scale systems deployed.
- ◆ 1990-1994: Shoddy measurements and questionable performance claims; faults generally ignored.
- ◆ 1994-1998: Numerous firms failed; government agencies cut funds.
- ◆ 1998-2002: Reassessment.
- ◆ 2002-2009: Recovery.
- ◆ 2010: 1 Pflop/s (10^{15} floating-point operations per second) demonstrated on a few large scientific computations.
- ◆ 2011: Heterogeneous architectures introduced; researchers and government agencies set sights on exascale computing.
- ◆ 2013: A broad international community is engaged in high-end computing.

Have lessons been learned? Or are we slipping back to hype and distortion?

Parallel system performance practices (1991)

- ◆ Performance results on small-sized parallel systems were linearly scaled to full-sized systems.
 - Example: 8,192-CPU results were linearly scaled to 65,536-CPU results, simply by multiplying by eight.
 - Rationale: “We can’t afford a full-sized system.”
 - Sometimes this was done without any clear disclosure in the paper or presentation.

Parallel system performance practices (1991), continued

- ◆ Highly tuned parallel implementations were compared with untuned implementations on other systems.
 - In comparisons of distributed memory systems with vector systems, often little or no effort was made to tune the vector code.
 - This was the case even for comparisons between SIMD parallel systems and vector systems -- here the SIMD code could have been converted rather easily to efficient vector code, but typically this was not done.

Parallel system performance practices (1991), continued

- ◆ Inefficient algorithms were employed, requiring many more operations, which resulted in artificially high Mflop/s rates.
 - Some scientists employed explicit PDE schemes for applications where implicit schemes were known to be much better.
 - One paper described doing a 3D discrete Fourier transform by direct evaluation of the defining formula, rather than by using a fast Fourier transform (i.e., $8n^2$ operations rather than $5n \log_2 n$, where n is the total number of points).

Parallel system performance practices (1991), continued

- ◆ Performance rates on 32-bit floating-point data on one system were compared with rates on 64-bit data on another system.
 - Using 32-bit data instead of 64-bit data effectively doubles data bandwidth, thus yielding artificially high performance rates.
 - Some computations can be done safely with 32-bit floating-point arithmetic, but many cannot.
 - In some emerging applications, even 64-bit (15-digit) floating-point arithmetic is not enough – many more digits are required.

Parallel system performance practices (1991), continued

- ◆ In some cases, performance experiments claimed in published papers *were not actually performed*. Example:
 - Abstract of published paper:

“The current Connection Machine implementation runs at 300-800 Mflop/s on a full [64K] CM-2, or at the speed of a single processor of a Cray-2 on 1/4 of a CM-2.”
 - Excerpt from text:

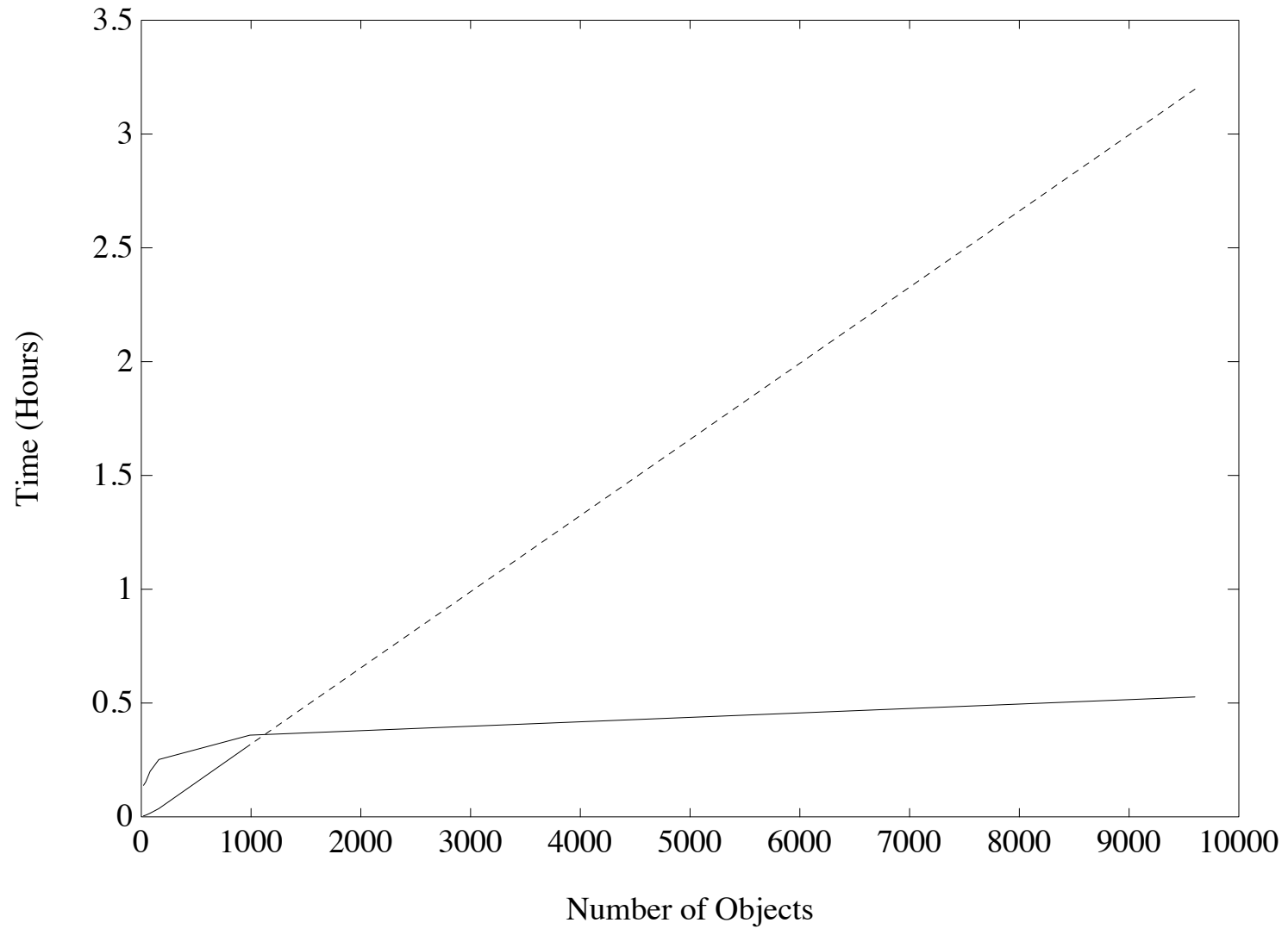
“This computation requires 568 iterations (taking 272 seconds) on a 16K Connection Machine.”

In other words, the computation was run on a 16K system, not on a 64K system. The figures cited in the Abstract were merely multiplied by four.
 - Another excerpt from text:

“In contrast, a Convex C210 requires 909 seconds to compute this example. Experience indicates that for a wide range of problems, a C210 is about 1/4 the speed of a single processor Cray-2, ...”

In other words, the computation mentioned in the Abstract was not actually run on a Cray-2. Instead, it was run on a Convex system, and a questionable scaling factor was used to produce the Cray-2 rate.

Performance plot A



Data for plot A

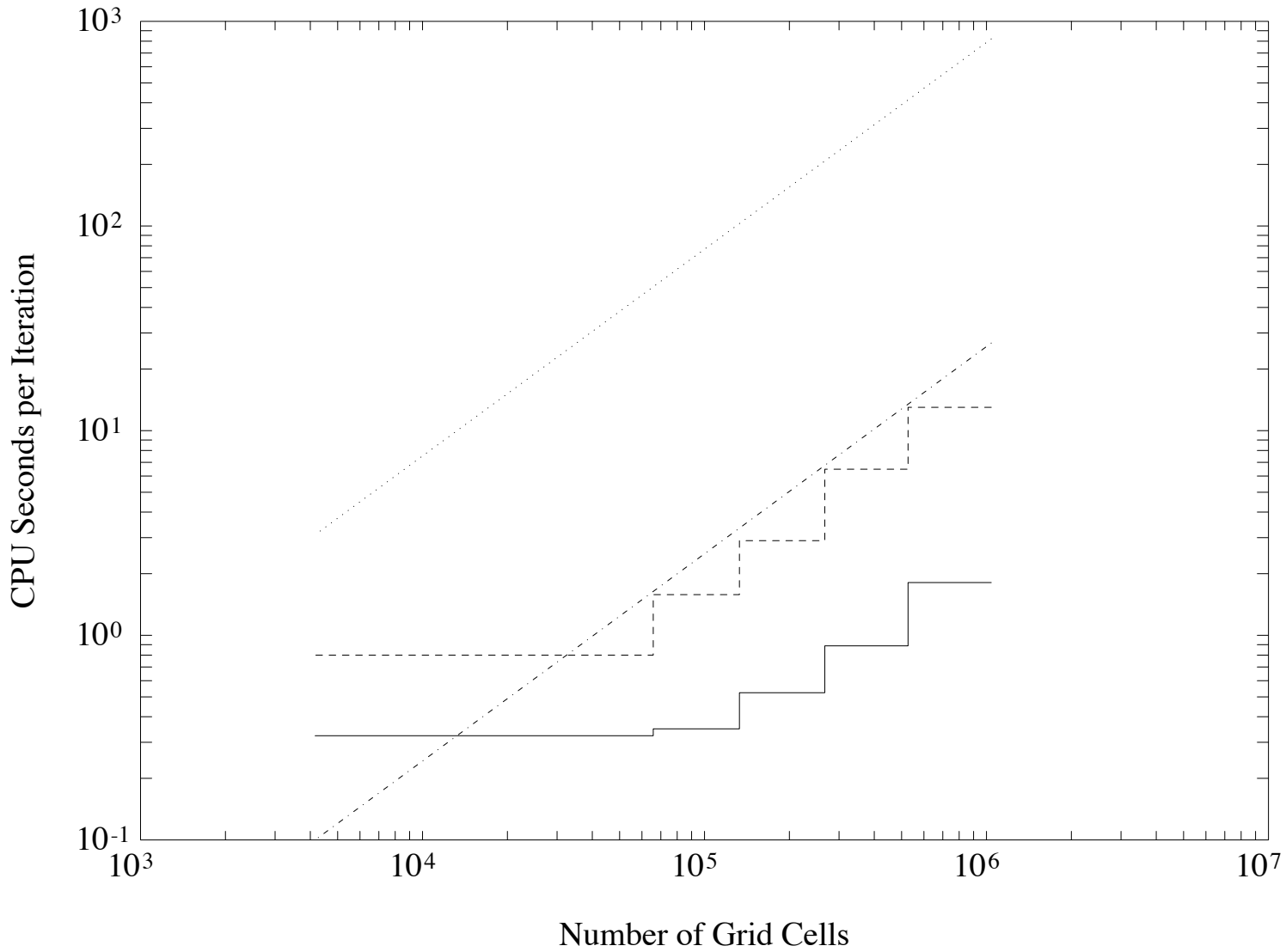
Problem size (x axis)	Parallel system run time	Vector system run time
20	8:18	0:16
40	9:11	0:26
80	11:59	0:57
160	15:07	2:11
990	21:32	19:00
9600	31:36	3:11:50*

Details in text of paper:

- ◆ In last entry, the 3:11:50 figure is an “estimate.”
- ◆ The vector system code is “not optimized.”

Note that the vector system performance is better in each run, except for the last (estimated) entry.

Performance plot B



Facts for plot B

- ◆ 32-bit performance rates on a parallel system are compared with 64-bit performance on a vector system.
- ◆ Parallel system results are linearly extrapolated to a full-sized system from a small system (only 1/8 size).
- ◆ The vector version of code is “unvectorized.”
- ◆ The vector system “curves” are straight lines – i.e., they are linear extrapolations from a single data point.

Conclusion: Most likely, only a handful of the points in these graphs represent real timings.

Parallel system performance practices (1991): other facts

- ◆ The examples in my files were written by professional scientists, and most were published in peer-reviewed journals and conference proceedings.
- ◆ One example is from an award-winning paper.
- ◆ In some cases, researchers accepted free computer time or other benefits from vendors, but did not disclose this fact in their papers.

DHB's "Twelve Ways to Fool the Masses" from Supercomputing Review, 1991

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.
6. Compare your results against scalar, unoptimized code on conventional systems.
7. When direct run time comparisons are required, compare with an old code on an obsolete system.
8. If Mflop/s 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 Mflop/s 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.

Technology

Measuring How Fast Computers Really Are

By JOHN MARKOFF

IN the world of scientific and technical computing, everyone agrees that computer speeds are increasing at a geometric rate. But measuring that speed is a vexing task. Rival supercomputer and work station manufacturers are prone to hype, choosing the performance figures that make their own machines look best.

"It's like the Wild West," said David J. Kuck, of the Center for Supercomputing Research and Development at the University of Illinois. "They say whatever they want to."

In fact, said David H. Bailey, a scientist at the National Aeronautics and Space Administration, "It's not really to the point of widespread fraud, but if people aren't a little more circumspect, the entire field could start to get a bad name."

The matter is complicated by a new generation of computers that have dozens, or even thousands, of separate processors. These parallel computers split problems into small parts and solve them simultaneously to reach greater speeds.

As a result, dozens of programs for determining benchmarks — measurements of computer speed — have been developed by scientists at universities and in government agencies. Some are based on how long a computer takes to solve a certain set of equations, while more sophisticated benchmarks attempt to match the operations re-

quired by real-world programs. But each benchmark generally measures only a single aspect of computer performance.

Just as a car buyer might buy a vehicle with the highest E.P.A. gas mileage rating for the price, a computer buyer could use benchmarks in deciding which machine to buy. But like their counterparts in the auto business, computer makers would do well to remind customers, "Your mileage may vary." The industry has no independent organization, analogous to the Environmental Protection Agency, to establish a single standard.

The proliferation of benchmarks is particularly problematic among the fastest scientific machines, where more than a dozen start-up companies compete to sell to university, corporate and Government laboratories.

These machines sell for hundreds of thousands of dollars or more, and the sale of only a few can mean success for a company. Supercomputers and smaller scientific work stations work on problems ranging from designing pharmaceuticals and weapons to weather modeling and the simulated crashing of automobiles.

Uneasy about the tendency for manufacturers to cite inflated claims, Mr. Bailey of NASA wrote a tongue-in-cheek indictment of performance claims for Supercomputing Review magazine in August. Titled "Twelve Ways to Fool the Masses When Giving Performance Results on Parallel Computers," it pokes fun at the tendency of computer mak-

Different Benchmarks, Different

The six fastest computers according to various benchmark point operations per second. Slalom, the only one accomplished in a set amount of time. The Perfect

LINPACK

Cray Y-MP/16	403
NEC SX-3/14	314
Cray Y-MP/832	275
Fujitsu VP2600/10	249
Cray X-MP/416	178
Cray 2S/4-128	129

Sources: Oak Ridge National Laboratory, Supercomputing Review, University of Illinois, University of Tennessee

ers to play fast and loose with speed claims.

It is common practice to "tune" computers and software to score better on benchmarks. "I know of a couple of companies who have full-time people, and all they do is optimize programs to achieve better benchmark results," said Gary Smaby, president of the Smaby Group, a consulting and market research firm in Minneapolis.

Such optimization is permissible under the rules established by benchmark designers to insure that computer makers can extract the full capability from their systems.

But some manufacturers go further and insert modules called "recognizers" into their compilers — software that translates a

Excerpts from NY Times article

“Rival supercomputer and work station manufacturers are prone to hype, choosing the performance figures that make their own systems look better.”

“It’s not really to the point of widespread fraud, but if people aren’t somewhat more circumspect, it could give the field a bad name.”

Fast forward to 2013: Some new ways to fool the masses

1. Cite performance rates for a run with only one processor core active in a shared-memory multi-core node. For example, cite performance on 1024 cores, even though the code was run on 1024 nodes, wasting 15 out of 16 cores on each node.
2. Cite performance rates only for a core algorithms (such as FFT or LU decomposition), even though the paper mentions one or more full-scale applications that were done on the system.
3. List only the best performance figure in the paper, even though the run was made numerous times.
4. Employ special hardware, operating system or compiler settings that are not appropriate for real-world usage.
5. Define “scalability” as successful execution on a large number of CPUs, regardless of performance.

Gerhard Willein's 14 new tricks

1. Report speedup instead of absolute performance.
2. Slow down code execution.
3. The log scale is your friend.
4. Quietly employ weak scaling to show off.
5. Instead of performance, plot absolute runtime versus CPU count.
6. Ignore affinity and topology issues.
7. Be creative when comparing scaled performance.
8. Impress your audience with awe-inspiring accuracy.
9. Boast massive speedups with accelerators.
10. Always emphasize the “interesting” part of your work.
11. Show data! Plenty. And then some.
12. Redefine “performance” to suit your needs.
13. If they get you cornered, blame it all on OS jitter.
14. Secretly use fancy hardware setups and software tricks.

-- Gerhard Willein, University of Erlangen, Germany

Source: <http://blogs.fau.de/hager/category/fooling-the-masses/>

Benchmarks help prevent abuse

- ◆ Benchmark tests must be well-designed, rigorous and scalable.
- ◆ Benchmark codes are problematic, because architectures and languages change, but “reference implementations” are useful.
- ◆ Well-thought-out and well-enforced “ground rules” are essential.
- ◆ A rational scheme must be provided for calculating performance rates.
- ◆ Tests must be specified to validate the correctness of the results.
- ◆ A repository of results must be maintained.

Examples:

- ◆ The NAS Parallel Benchmarks (still useful, but a bit dated).
- ◆ The LBNL-UCB “Torch” Reference Kernels. See:
http://crd.lbl.gov/~dhbailey/dhbpapers/dwarfs09_intro.pdf

General guidelines to prevent abuse

- ◆ Direct comparisons of run times on real applications are preferred.
- ◆ If results are presented for a well-known benchmark, established ground rules must be followed.
- ◆ Only actual performance results should be presented, not projections or extrapolations (unless very clearly disclosed and justified).
- ◆ Performance figures should be based on comparable levels of tuning.
- ◆ Mflop/s, Gflop/s, Tflop/s rates should be computed from operation counts based on the best practical serial algorithms.
- ◆ When computing parallel speedup figures, the denominator rate should be based on an efficient single-processor implementation.
- ◆ Any ancillary information that would affect the interpretation of the results should be fully disclosed (e.g., the use of 32-bit instead of 64-bit data, etc.).
- ◆ Special care should be taken for figures and graphs.
- ◆ Whenever possible, full background information should be provided: algorithms, hardware and software configuration, language, compiler flags, tuning, timing method, basis for operation counts, etc.

This talk is available at:

<http://www.davidhbailey.com/dhbtalks/dhb-12ways-2013.pdf>