Big data and its sins

David H. Bailey
http://www.davidhbailey.com
Lawrence Berkeley National Laboratory (retired)
Computer Science Department, University of California, Davis

November 13, 2015

Big data science: DANGER AHEAD

Supercomputers operating on big data can generate nonsense faster than ever before!

Key concerns:

- Are the results statistically sound?
- Are the results numerically reliable?
- Have the results been validated using independent rigorous tests?
- Are the algorithms, data sources and processing methods well documented?

Reproducibility crises in physics and cosmology

- In 2011, an international team of researchers at the Gran Sasso Laboratory in Italy announced that neutrinos had exceeded the speed of light, thus directly challenging Einstein's relativity. However, after months of careful checking, a subtle flaw was found in the measurement apparatus (a major embarrassment).
- In 2013, CERN researchers confirmed the discovery of the long-sought Higgs boson. But more recently, scientists have raised questions as to whether the particle discovered is really the Higgs – it might be some other particle or particles masquerading as the Higgs; additional research studies are required.
- In March 2014, researchers announced with considerable fanfare that they had detected the fingerprint of the long-hypothesized inflationary epoch, a tiny fraction after the big bang. Sadly, within a few weeks critics pointed out that their experimental results might well be due to dust in the Milky Way, pending better data.

Reproducibility crises in biomedicine, psychology, economics, finance, autos

- In 2011, Bayer researchers reported that they were able to reproduce only 17 of 67 pharma studies.
- In 2012, Amgen researchers reported that they were able to reproduce only 6 of 53 cancer studies.
- In August 2015, the Reproduciblity Project in Virginia reported that they were able to reproduce only 39 of 100 psychology studies.
- In September 2015, the U.S. Federal Reserve was able to reproduce only 29 of 67 economics studies.
- In 2014-2015, "backtest overfitting" emerged as a major problem in computational finance.

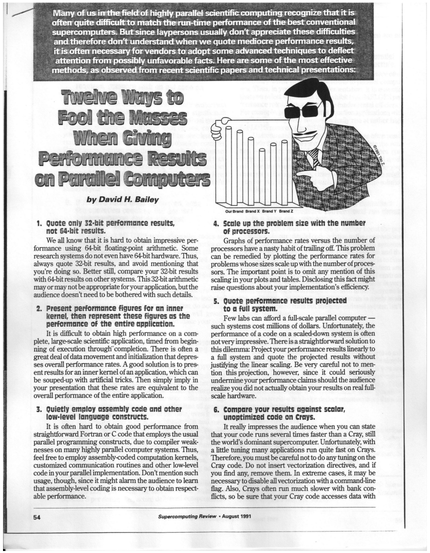
Reproducibility Project staff

Credit: NY Times

- In March 2014, West Virginia researchers reported that they were unable to reproduce Volkswagen's claimed emission figures. This has now exploded into a major international scandal.

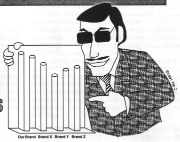
Parallel computing, 1991

- Many new parallel systems were offered; users were excited about potential.
- Each vendor claimed theirs was best, citing one or two selected applications.
- There were few standard benchmarks or testing methodologies — mostly Livermore Loops and original Linpack-100.
- Overall, the level of rigor and peer review in the field was disappointingly low.
- In 1991 DHB published a humorous essay Twelve Ways to Fool the Masses, poking fun at some of the abuses (authored by DHB, but with contributions from the NPB team).



Many of us in the field of highly parallel scientific computing recognize that it is often quite difficult to match the run-time performance of the best conventional supercomputers. But since laypersons usually don't appreciate these difficulties and therefore don't understand when we quote mediocre performance results, it is often necessary for vendors to adopt some advanced techniques to deflect attention from possibly unfavorable facts. Here are some of the most effective methods, as observed from recent scientific papers and technical presentations:

Twelve Ways to Fool the Masses When Giving Performance Results on Parallel Computers



by David H. Bailey

1. Quote only 32-bit performance results, not 64-bit results.

We all know that it is hard to obtain impressive performance using 64-bit floating-point arithmetic. Some research systems do not even have 64-bit hardware. Thus, always quote 32-bit results, and avoid mentioning that you're doing so. Better still, compare your 32-bit results with 64-bit results on other systems. This 32-bit arithmetic may or may not be appropriate for your application, but the audience doesn't need to be bothered with such details.

2. Present performance figures for an inner kernel, then represent these figures as the performance of the entire application.

It is difficult to obtain high performance on a complete, large-scale scientific application, timed from beginning of execution through completion. There is often a great deal of data movement and initialization that depresses overall performance rates. A good solution is to present results for an inner kernel of an application, which can be souped-up with artificial tricks. Then simply imply in your presentation that these rates are equivalent to the overall performance of the entire application.

3. Quietly employ assembly code and other low-level language constructs.

It is often hard to obtain good performance from straightforward Fortran or C code that employs the usual parallel programming constructs, due to compiler weaknesses on many highly parallel computer systems. Thus, feel free to employ assembly-coded computation kernels, customized communication routines and other low-level code in your parallel implementation. Don't mention such usage, though, since it might alarm the audience to learn that assembly-level coding is necessary to obtain respectable performance.

4. Scale up the problem size with the number of processors.

Graphs of performance rates versus the number of processors have a nasty habit of trailing off. This problem can be remedied by plotting the performance rates for problems whose sizes scale up with the number of processors. The important point is to omit any mention of this scaling in your plots and tables. Disclosing this fact might raise questions about your implementation's efficiency.

5. Quote performance results projected to a full system.

Few labs can afford a full-scale parallel computer — such systems cost millions of dollars. Unfortunately, the performance of a code on a scaled-down system is often not very impressive. There is a straightforward solution to this dilemma: Project your performance results linearly to a full system and quote the projected results without justifying the linear scaling. Be very careful not to mention this projection, however, since it could seriously undermine your performance claims should the audience realize you did not actually obtain your results on real full-scale hardware.

6. Compare your results against scalar, unoptimized code on Crays.

It really impresses the audience when you can state that your code runs several times faster than a Cray, still the world's dominant supercomputer. Unfortunately, with a little tuning many applications run quite fast on Crays. Therefore, you must be careful not to do any tuning on the Cray code. Do not insert vectorization directives, and if you find any, remove them. In extreme cases, it may be necessary to disable all vectorization with a command-line flag. Also, Crays often run much slower with bank conflicts, so be sure that your Cray code accesses data with

54 *Supercomputing Review* • August 1991

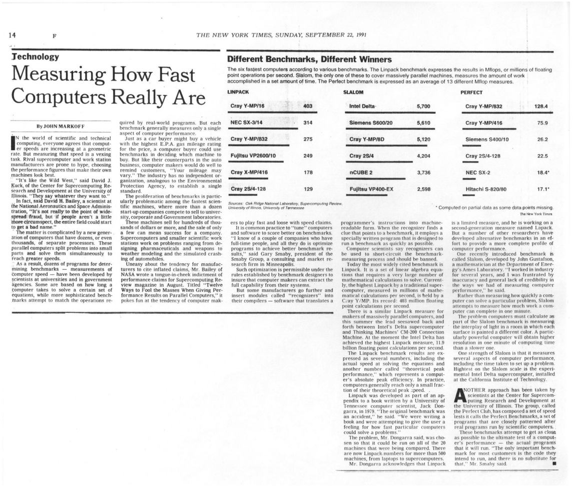
Twelve ways to fool the masses (paraphrased)

1. Quote 32-bit performance results, not 64-bit results, but don't mention this in paper.
2. Present performance figures for an inner kernel, 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.
5. Quote performance results projected to a full system.
6. Compare your results against scalar, unoptimized code on conventional systems.
7. When run times are compared, compare with an old code on an obsolete system.
8. Base Mflop/s rates on the operation count of the parallel implementation, instead of the best practical serial algorithm.
9. Quote performance as 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 discuss performance.

NY Times (22 Sep 1991): "Measuring How Fast Computers Really Are"

Excerpts:

- "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." [quoting David J. Kuck of UIUC].
- "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." [quoting DHB].



14 F THE NEW YORK TIMES, SUNDAY, SEPTEMBER 22, 1991

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 required 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 makers 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 programmer's instructions into machine-readable form. When the recognizer finds a clue that points to a benchmark, it employs a specially written program that is designed to run a benchmark as quickly as possible.

Computer scientists say recognizers can be used to short-circuit the benchmark-measuring process and should be banned.

Today the most widely cited benchmark is Linpack. It is a set of linear algebra equations that requires a very large number of mathematical calculations to solve. Currently, the highest Linpack by a traditional supercomputer, measured in millions of mathematical calculations per second, is held by a Cray Y/MP. Its record: 403 million floating point calculations per second.

There is a similar Linpack measure for makers of massively parallel computers, and this summer the lead seesawed back and forth between Intel's Delta supercomputer and Thinking Machines' CM-200 Connection Machine. At the moment the Intel Delta has achieved the highest Linpack measure, 11.9 billion floating point calculations per second.

The Linpack benchmark results are expressed as several numbers, including the actual speed at solving the equations and another number called "theoretical peak performance," which represents a computer's absolute peak efficiency. In practice, computers generally reach only a small fraction of their theoretical peak speed.

Linpack was developed as part of an appendix to a book written by a University of Tennessee computer scientist, Jack Dongarra, in 1979. "The original benchmark was an accident," he said. "We were writing a book and were attempting to give the user a feeling for how fast particular computers could solve a problems."

The problem, Mr. Dongarra said, was chosen so that it could be run on all of the 20 machines that were being compared. There are now Linpack numbers for more than 500 machines, from laptops to supercomputers.

Mr. Dongarra acknowledges that Linpack is a limited measure, and he is working on a second-generation measure named Lapack. But a number of other researchers have developed alternative benchmarks in an effort to provide a more complete profile of computer performance.

One recently introduced benchmark is called Slalom, developed by John Gustafson, a mathematician at the Department of Energy's Ames Laboratory. "I worked in industry for several years, and I was frustrated by inaccuracy and general lack of credibility in the ways we had of measuring computer performance," he said.

Rather than measuring how quickly a computer can solve a particular problem, Slalom attempts to measure how much work a computer can complete in one minute.

The problem computers must calculate as part of the Slalom benchmark is measuring the interplay of light in a room in which each surface is painted a different color. A particularly powerful computer will obtain higher resolution in one minute of computing time than a slower one.

One strength of Slalom is that it measures several aspects of computer performance, including the time taken to set up a problem. Hightest on the Slalom scale is the experimental Intel Delta supercomputer, installed at the California Institute of Technology.

ANOTHER approach has been taken by scientists at the Center for Supercomputing Research and Development at the University of Illinois. The group, called the Perfect Club, has composed a set of speed tests it calls the Perfect Benchmarks, a set of programs that are closely patterned after real programs run by scientific computers.

These benchmarks attempt to get as close as possible to the ultimate test of a computer's performance — the actual programs that it will run. "The only important benchmark for most customers is the code they intend to run, and there is no substitute for that," Mr. Smaby said. ■

Different Benchmarks, Different Winners

The six fastest computers according to various benchmarks. The Linpack benchmark expresses the results in Mflops, or millions of floating point operations per second. Slalom, the only one of these to cover massively parallel machines, measures the amount of work accomplished in a set amount of time. The Perfect benchmark is expressed as an average of 13 different Mflop measures.

LINPACK		SLALOM		PERFECT	
Cray Y-MP/16	403	Intel Delta	5,700	Cray Y-MP/832	128.4
NEC SX-3/14	314	Siemens S600/20	5,610	Cray Y-MP/416	75.9
Cray Y-MP/832	275	Cray Y-MP/8D	5,120	Siemens S400/10	26.2
Fujitsu VP2600/10	249	Cray 2S/4	4,204	Cray 2S/4-128	22.5
Cray X-MP/416	178	nCUBE 2	3,736	NEC SX-2	18.4*
Cray 2S/4-128	129	Fujitsu VP400-EX	2,598	Hitachi S-820/80	17.1*

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

* Computed on partial data as some data points missing.

The New York Times

Examples of abuses: Scaling performance results to full-sized system

In some published papers and conference presentations, runs were performed on smaller systems, then performance rates were linearly scaled to full-sized systems, in some cases without even clearly disclosing this fact.

Example: 8,192-CPU performance results were linearly scaled to 65,536-CPU results, simply by multiplying by eight.

Typical excuse: "We can't afford a full-sized system."

- D. H. Bailey, "Misleading performance reporting in the supercomputing field," *Scientific Programming*, vol. 1., no. 2 (Winter 1992), 41–151.

Using inefficient algorithms on highly parallel systems

In many cases, inefficient algorithms were employed for highly parallel implementations, requiring many more operations, thus producing artificially high speedup figures and Mflop/s rates. Examples:

- Some researchers cited parallel PDE performance based on explicit schemes, for problems where implicit schemes were known to be significantly more efficient.
- One paper cited performance for computing a 3D discrete Fourier transform by direct evaluation of the defining formula ($8n^2$ operations), rather than by using a fast Fourier transform ($5n \log_2 n$).

Both examples violate a basic rule of parallel computing, namely to base parallel implementations and operation counts (when computing Mflop/s or Gflop/s rates) on the best practical serial algorithm.

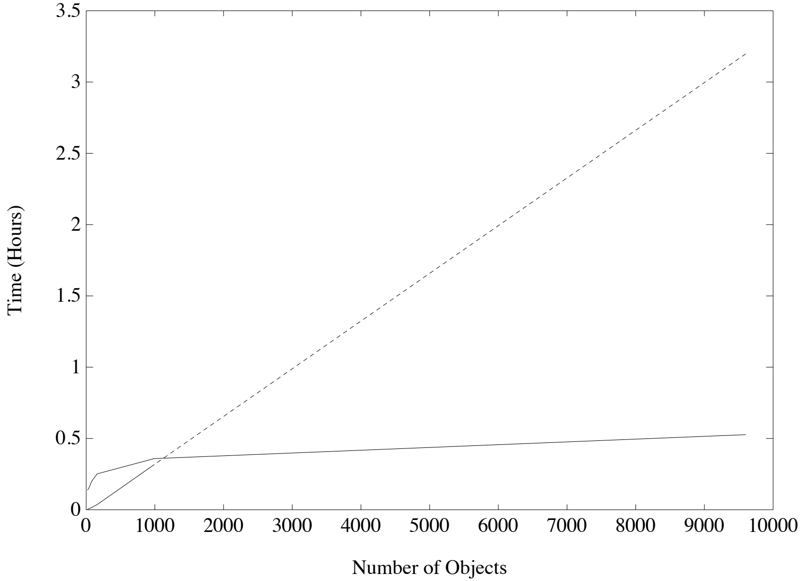
Typical excuse: Other algorithms are “more appropriate” for our parallel system.

Abstract versus details in paper

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.
- 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 actually run on a Convex system, and a rule-of-thumb scaling factor was used to produce the Cray-2 rate.

Performance plot: parallel run times (lower) vs vector (upper)



Data for performance plot

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 parallel system is actually slower than the vector system for all cases, except for the last (estimated) entry. Also, except for the last entry, all real data in the graph is in the lower left corner. A log-log plot should have been used instead.

Origins of the NAS Parallel Benchmarks (NPB)

- In 1991, a team of 12 researchers at the Numerical Aerodynamic Simulation (NAS) facility (now known as the NASA Advanced Supercomputing facility) formulated the "NAS Parallel Benchmarks."
- The original plan was to be a basis for an upcoming supercomputer procurement for NAS.
- A central goal of the NPB was to test algorithms of importance in computational aeronautics.
- However, the NPB team recognized from the beginning that the benchmarks should be designed to have wider usage — hopefully to help clear the hype and confusion in the field.

The resulting paper (below) has now been awarded the 2015 "Test of Time Award" from the ACM / IEEE Supercomputing Conference.

- D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. .A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan and S. Weeratunga, "The NAS Parallel Benchmarks: Summary and preliminary results," *Proceedings of Supercomputing 1991*, ACM/IEEE Computer Society, 1991, 158–165.

2015: New ways to fool the masses in HPC

- Cite performance rates for a run with only one processor core active in a shared-memory multi-core node, producing artificially inflated performance (since there is no shared memory interference) and wasting resources (since most cores are idle).
  - Example: Cite performance on “1024 cores,” even though the code was run on 1024 multicore nodes, one core per node, with 15 out of 16 cores idle on each node.
- Claim that since one is using a graphics processing unit (GPU) system, the most efficient algorithms must be discarded in favor of “more appropriate” algorithms (recall the experience with parallel algorithms).
- Run the test code many times, but only include the best performance rate in the paper (recall the experience of recent pharmaceutical trials).
- Employ special hardware, operating system or compiler settings that are not appropriate for real-world production usage (recall the recent Volkswagen scandal).

ICERM report on numerical reproducibility

Numerical round-off error and numerical differences are greatly magnified as computational simulations are scaled up to run on highly parallel systems. As a result, it is increasingly difficult to determine whether a code has been correctly ported to a new system, because computational results quickly diverge from standard benchmark cases. And it is doubly difficult for other researchers, using independently written codes and distinct computer systems, to reproduce published results.

- 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," 2012, `http://www.davidhbailey.com/dhbpapers/icerm-report.pdf`.

Numerical reproducibility

Many applications routinely use either 32-bit or 64-bit IEEE arithmetic, and employ fairly simple algorithms, assuming that all is well. But problems can arise:

- Large-scale, highly parallel simulations, running on systems with hundreds of thousands or millions of processors greatly magnify numerical sensitivities.
- Certain applications with highly ill-conditioned linear systems are particularly vulnerable.
- Large summations, especially those involving $+/$- terms and cancellations, can produce errors.
- Long-time, iterative simulations (such as molecular dynamics or climate models) are notoriously non-reproducible.
- Computations to resolve small-scale phenomena often require high precision to produce numerically reproducible results.
- Studies in computational physics or experimental mathematics often require huge precision levels.

Analysis of collisions at the Large Hadron Collider

- The 2012 discovery of the Higgs boson at the ATLAS experiment in the LHC relied crucially on the ability to track charged particles with exquisite precision (10 microns over a 10m length) and high reliability (over 99% of roughly 1000 charged particles per collision correctly identified).
- Software: 5 millions line of C++ and Python code, developed by roughly 2000 physicists and engineers over 15 years.
- Recently, in an attempt to speed up the calculation, researchers found that merely changing the underlying math library resulted in some collisions being missed or misidentified.

Questions:

- How serious are these numerical difficulties?
- How can they be tracked down?
- How can the library be maintained, producing numerically reliable results?

Approaches to solve accuracy and numerical reproducibility problems

- Employ an expert numerical analyst to re-examine every algorithm employed in the computation to ensure that only the most stable known schemes are being used.
- Carefully analyze every step of the computation to ascertain the level of numerical sensitivity at each step.
- Employ interval arithmetic for large portions of the application (which greatly increases run time and code complexity).
- Employ some “smart” tools to help identify the numerically sensitive parts of an application, then use higher-precision arithmetic arithmetic if needed to fix them.

In most cases, the last item is the only practical solution.

Numerical analysis expertise among U.C. Berkeley graduates

Of the 2010 U.C. Berkeley graduating class, 870 were in disciplines likely to require technical computing:

- Division of Mathematical and Physical Sciences (Math, Physics, Statistics).
- College of Chemistry.
- College of Engineering (including Computer Science).

Other fields (not counted) that will likely involve significant computing:

- Biology, geology, medicine, economics, psychology, sociology.

Enrollment in numerical analysis courses:

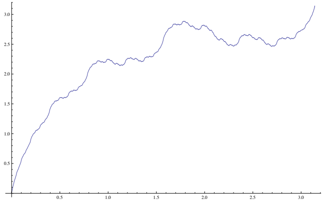
- Math 128A (Introductory numerical analysis required of math majors): 219.
- Math 128B (A more advanced course, required to do serious work): 24.

Conclusion: Fewer than 2% of Berkeley graduates who will do technical computing have had rigorous training in numerical analysis.

Enhancing reproducibility with selective usage of high-precision arithmetic

Problem: Find the arc length of the irregular function $g(x) = x + \sum_{0 \le k \le 10} 2^{-k} \sin(2^k x)$, over the interval $(0, \pi)$ (using 10^7 abscissa points).

- If this computation is done with ordinary double precision arithmetic, the calculation takes 2.59 seconds and yields the result 7.073157029008510.
- If it is done using all double-double arithmetic (31-digit accuracy), it takes 47.39 seconds seconds and yields the result 7.073157029007832 (correct to 15 digits).
- But if only the summation is changed to double-double, the result is identical to the double-double result (to 15 digits), yet the computation only takes 3.47 seconds.



Graph of $g(x) = x + \sum_{0 \le k \le 10} 2^{-k} \sin(2^k x)$, over $(0, \pi)$.

U.C. Berkeley's CORVETTE project and the "Precimonious" tool

Objective: Develop software facilities to find and ameliorate numerical anomalies in large-scale computations:

- Test the level of numerical accuracy required for an application.
- Delimit the portions of code that are inaccurate.
- Search the space of possible code modifications.
- Repair numerical difficulties, including usage of high-precision arithmetic.
- Navigate through a hierarchy of precision levels (32-bit, 64-bit, 80-bit or higher as needed).

The current version of this tool is known as "Precimonious."

- C. Rubio-Gonzalez, C. Nguyen, H. D. Nguyen, J. Demmel, W. Kahan, K. Sen, D. H. Bailey and C. Iancu, "Precimonious: Tuning assistant for floating-point precision," *Proceedings of SC13*, Nov 2013.

Precimonious in action for the arc length computation

Original code:

```
int main() {
  int i, n = 1000000;
  long double h, t1, t2, dppi;
  long double s1;
  ...
  for (i = 1; i <= n; i++) {
    t2 = fun (i * h);
    s1 = s1 + sqrt (h*h + (t1-t2)*(t1-t2));
    t1 = t2;
  }
  return 0;
}
```

Automatically tuned code:

```
int main() {
  int i, n = 1000000;
  double h, t1, t2, dppi;
  long double s1;
  ...
  for (i = 1; i <= n; i++) {
    t2 = fun (i * h);
    s1 = s1 + sqrt (h*h + (t1-t2)*(t1-t2));
    t1 = t2;
  }
  return 0;
}
```

The tuned code produces the same answers, to 15-digit accuracy, yet runs faster.

John Gustafson's new "unum" system for floating-point

Gustafson proposes a radical new definition of floating-point arithmetic, based on unums (short for "universal numbers"), a flexible format that contains a "ubit" to designate that the true number lies between two adjacent binary floats.

Example: Avogadro's constant $= 6.022 \times 10^{23}$ is (29 bits total):

sign	exp	frac	ubit	exp size	frac size
0	11001101	111111100001	1	111	1011

Advantages:

- Obeys commutative law, transitive law and other laws of algebra.
- No rounding error, overflow, underflow, negative zero.
- Consistent treatment of positive and negative infinity and NaN.
- Safe to parallelize; portable between systems.
- Often requires fewer bits than conventional IEEE 64-bit arithmetic.

Sample problem (due to Jean-Michel Muller)

Define:

$$\begin{aligned} E(z) &= (e^z - 1)/z, \quad E(0) = 1, \\ Q(x) &= |x - \sqrt{x^2+1}| - 1/(x + \sqrt{x^2+1}), \\ H(x) &= E(Q^2(x)). \end{aligned}$$

Problem: Compute $H(x)$ for $x = 15, 16, 17, 9999$.

- IEEE 32-bit: $(0, 0, 0, 0)$.
- IEEE 64-bit: $(0, 0, 0, 0)$.
- IEEE 128-bit: $(0, 0, 0, 0)$.
- Unum (correct): $(1, 1, 1, 1)$.

Another sample problem (due to Siegfried Rump)

Define

$$F(x, y) = (333 + 3/4)y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + (5 + 1/2)y^8 + x/(2y)$$

Problem: Compute $F(77617, 33096)$.

- IEEE 32-bit: 1.172603.
- IEEE 64-bit: 1.1726039400531.
- IEEE 128-bit: 1.172603940053178.
- Unum (correct to 23 digits): $-0.827396059946821368141165\ldots$.

Unum: Current status and plans

- Gustafson's book has been published by CRC Press (see below).
- A full implementation is nearly complete in C.
- Much testing and analysis will be done.
- Will require a flexible working precision level.
- Hardware implementation?

For additional details:

- John Gustafson, *The End of Error: Unum Computing*, CRC Press, 2015.

Aren't 64 bits enough?

There has been considerable resistance in the scientific computing community to the notion that more than 64-bit arithmetic is not only useful, but may be essential in some scientific computations.

Many believe:

- Physical reality fundamentally does not require high precision, beyond, say, 64-bit IEEE arithmetic, for any computation.
- Numerical sensitivity problems are always due to the usage of inferior algorithms.
- Conversion costs are always too high, and the added computational costs are always too great.
- The usage of high-precision arithmetic is "cheating" or "sinful" (because it is too easy?).

Innocuous example where standard 64-bit precision is inadequate

Problem: Find a polynomial to fit the data $(1, 1048579, 16777489,$ $84941299, 268501249, 655751251, 1360635409, 2523398179,\ 4311748609)$ for arguments $0, 1, \cdots, 8$. The usual approach is to solve the linear system:

$$\begin{bmatrix} n & \sum_{k=1}^{n} x_k & \cdots & \sum_{k=1}^{n} x_k^n \\ \sum_{k=1}^{n} x_k & \sum_{k=1}^{n} x_k^2 & \cdots & \sum_{k=1}^{n} x_k^{n+1} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{k=1}^{n} x_k^n & \sum_{k=1} x_k^{n+1} & \cdots & \sum_{k=1}^{n} x_k^{2n} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} \sum_{k=1}^{n} y_k \\ \sum_{k=1}^{n} x_k y_k \\ \vdots \\ \sum_{k=1}^{n} x_k^n y_k \end{bmatrix}$$

A 64-bit computation (e.g., using Matlab, Linpack or LAPACK) fails to find the correct polynomial in this instance, even if one rounds results to nearest integer.

However, if Linpack routines are converted to use double-double arithmetic (31-digit accuracy), the above computation quickly produces the correct polynomial:

$$f(x) \;=\; 1 + 1048577x^4 + x^8 \;=\; 1 + (2^{20}+1)x^4 + x^8$$

Innocuous example, cont.

The result on the previous page can be obtained with double precision using Lagrange interpolation or the Demmel-Koev algorithm. But few scientists, outside of expert numerical analysts, are aware of these schemes.

Besides, even these schemes fail for higher-degree problems.

For example: $(1, 134217731, 8589938753, 97845255883, 549772595201, 2097396156251, 6264239146561, 15804422886323, 35253091827713, 71611233653971, 135217729000001, 240913322581691, 409688091758593)$
is generated by:

$$f(x) \;=\; 1 + 134217729x^6 + x^{12} \;=\; 1 + (2^{27} + 1)x^6 + x^{12}$$

Lagrange interpolation and Demmel-Koev fail for this problem, but a straightforward Linpack scheme, implemented with double-double arithmetic, works fine.

Free software for high-precision computation

1. ARPREC. Arbitrary precision, with numerous algebraic and transcendental functions. High-level interfaces for C++ and Fortran. `http://crd-legacy.lbl.gov/~dhbailey/mpdist`.
2. GMP. Low-level package for arbitrary precision integer and floating-point arithmetic. `http://gmplib.org`.
3. MPFR. C library for arbitrary-precision floating-point computations with exact rounding, based on GMP. `http://www.mpfr.org`.
4. MPFR++. High-level C++ interface to MPFR. `http://perso.ens-lyon.fr/nathalie.revol/software.html`.
5. MPFUN2015. Similar to ARPREC, but based on MPFR, with a Fortran interface. `http://crd-legacy.lbl.gov/~dhbailey/mpdist`.
6. QD. Performs "double-double" (31 digits) and "quad-double" (62 digits) arithmetic. High-level interfaces for C++ and Fortran-90. `http://crd-legacy.lbl.gov/~dhbailey/mpdist`.

Some applications where high precision is useful or essential

1. Planetary orbit calculations (32 digits).
2. Supernova simulations (32–64 digits).
3. Climate modeling (32 digits).
4. Coulomb n-body atomic system simulations (32–120 digits).
5. Schrodinger solutions for lithium and helium atoms (32 digits).
6. Electromagnetic scattering theory (32–100 digits).
7. Scattering amplitudes of fundamental particles (32 digits).
8. Discrete dynamical systems (32 digits).
9. Theory of nonlinear oscillators (64 digits).
10. The Taylor algorithm for ODEs (100–600 digits).
11. Ising integrals from mathematical physics (100–1000 digits).
12. Problems in experimental mathematics (100–50,000 digits).

- D. H. Bailey, R. Barrio, and J. M. Borwein, “High precision computation: Mathematical physics and dynamics,” *Applied Mathematics and Computation*, vol. 218 (2012), pg. 10106–10121.

Long-term planetary orbit calculations

Researchers have recognized for centuries that planetary orbits exhibit chaotic behavior:

> *"The orbit of any one planet depends on the combined motions of all the planets, not to mention the actions of all these on each other. To consider simultaneously all these causes of motion and to define these motions by exact laws allowing of convenient calculation exceeds, unless I am mistaken, the forces of the entire human intellect."* *[Isaac Newton, Principia, 1687]*

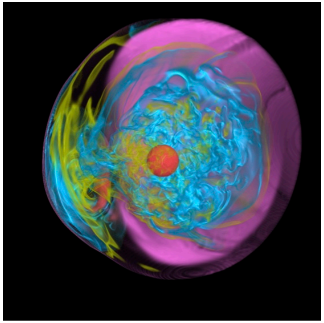
Long-term simulations of planetary orbits using double precision do fairly well for long periods, but then fail at certain key junctures.

Researchers have found that double-double or quad-double arithmetic is occasionally required to avoid severe inaccuracies, even if other techniques are employed to reduce numerical error.

- G. Lake, T. Quinn and D. C. Richardson, "From Sir Isaac to the Sloan survey: Calculating the structure and chaos due to gravity in the universe," *Proc. of the 8th ACM-SIAM Symp. on Discrete Algorithms*, 1997, pg. 1–10.

Supernova simulations

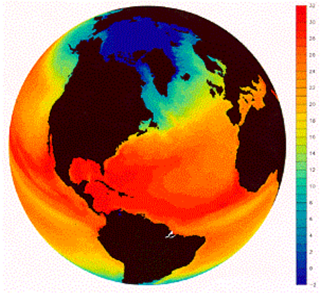
- Researchers at LBNL have used quad-double arithmetic to solve for non-local thermodynamic equilibrium populations of iron and other atoms in the atmospheres of supernovas.
- Iron may exist in several species, so it is necessary to solve for all species simultaneously.
- Since the relative population of any state from the dominant state is proportional to the exponential of the ionization energy, the dynamic range of these values can be very large, and cancellations may occur.
- Quad-double arithmetic (62 digits) was employed to reduce these errors.



P. H. Hauschildt and E. Baron, "The numerical solution of the expanding stellar atmosphere problem," *Journal Computational and Applied Mathematics*, vol. 109 (1999), pg. 41–63.

Climate modeling: High-precision for reproducibility

- Climate and weather simulations are fundamentally chaotic: if microscopic changes are made to the current state, soon the future state is quite different.
- In practice, computational results are altered even if minor changes are made to the code or the system.
- This numerical variation is a major nuisance for code maintenance.
- He and Ding found that by using double-double arithmetic in two key inner loops, most of this numerical variation disappeared.



Y. He and C. Ding, "Using accurate arithmetics to improve numerical reproducibility and stability in parallel applications," *Journal of Supercomputing*, vol. 18, no. 3 (Mar 2001), pg.

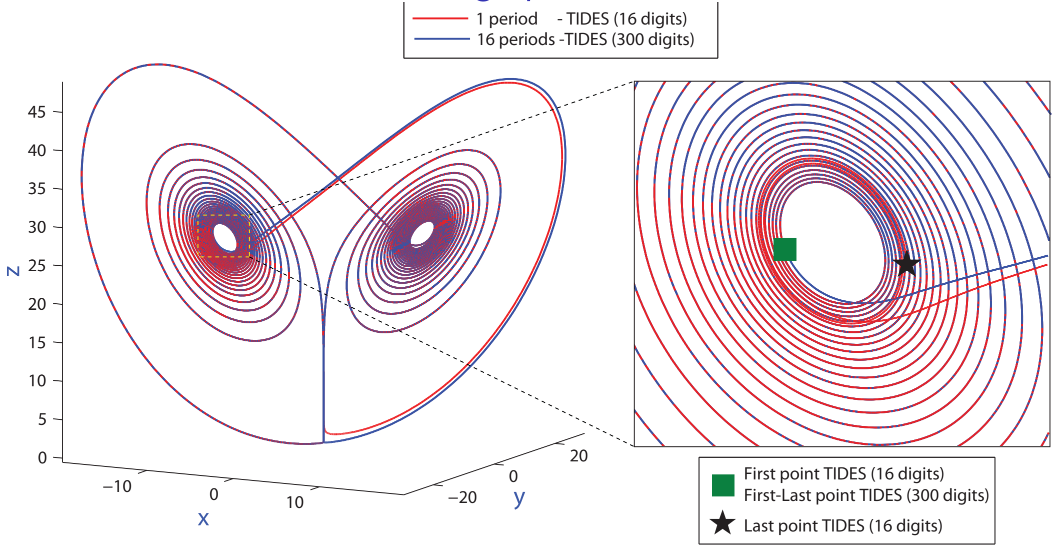
Coulomb n-body atomic system simulations

- Alexei Frolov of Queen's University in Canada has used high-precision arithmetic to solve a generalized eigenvalue problem that arises in Coulomb n-body interactions.
- Matrices are typically 5,000 x 5,000 and are very nearly singular.
- Computations typically involve massive cancellation, and high-precision arithmetic must be employed to obtain numerically reproducible results.
- Frolov has also computed elements of the Hamiltonian matrix and the overlap matrix in four- and five-body systems.
- These computations typically require 120-digit arithmetic.

Frolov: "We can consider and solve the bound state few-body problems ... beyond our imagination even four years ago."

- D. H. Bailey, R. Barrio, and J. M. Borwein, "High precision computation: Mathematical physics and dynamics," *Applied Mathematics and Computation*, vol. 218 (2012), pg. 10106–10121.

Taylor's method for ODEs with high-precision arithmetic



Numerical integration of the L25-R25 unstable orbit for the Lorenz model during 16 time periods using the TIDES code with 300 digits, and 1 time period using 15 digits.

Discovering new results in mathematics and mathematical physics

Methodology:

1. Compute various mathematical entities (limits, infinite series sums, definite integrals, etc.) to very high precision, typically 100–100,000 digits.
2. Use algorithms such as PSLQ to recognize these numerical values in terms of well-known mathematical constants.
3. When results are found experimentally, seek formal proofs of the discovered relations.

If mathematics describes an objective world just like physics, there is no reason why inductive methods should not be applied in mathematics just the same as in physics.
— Kurt Godel

Summary

- Numerical reproducibility is looming as a major challenge for large-scale scientific computations.
- One practical solution is to apply intellgient tools that judiciously employ higher-precision arithmetic for numerically sensitive portions of the code.
- Some interesting computations in experimental mathematics and mathematical physics require even more precision — hundreds or even many thousands of digits.
- Such computations are now feasible using modern high-performance computing systems, together with relatively easy-to-use software facilities.

This talk is available at
`http://www.davidhbailey.com/dhbtalks/dhb-bigdata-sins.pdf`.