Fooling the masses in 2017: Why the scientific computing community must still be vigilant

David H. Bailey
http://www.davidhbailey.com
Lawrence Berkeley National Lab (retired) and Univ. of California, Davis

May 12, 2017
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?
- Are performance results accurately and honestly reported?
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 and finance

- 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 Reproducibility 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.

These experiences have exposed at least one common flaw: Only publicizing the results of successful trials introduces a bias into the results.
Reproducibility in scientific computing

A December 2012 workshop on reproducibility in computing, held at Brown University in Rhode Island, USA, noted that

*Science is built upon the foundations of theory and experiment validated and improved through open, transparent communication. With the increasingly central role of computation in scientific discovery this means communicating all details of the computations needed for others to replicate the experiment.*

... The “reproducible research” movement recognizes that traditional scientific research and publication practices now fall short of this ideal, and encourages all those involved in the production of computational science ... to facilitate and practice really reproducible research.

ICERM Report: To aid in reproducibility, computing papers should document:

- A precise statement of assertions to be made in the paper.
- A statement of the computational approach, and why it constitutes a test of hypothesis.
- Complete statements of (or references to) every algorithm employed.
- Details of auxiliary software (both research and commercial software) used.
- Details of the test environment, including hardware, system software and the number of processors utilized.
- Details of data reduction and statistical analysis methods.
- Discussion of the adequacy of parameters such as precision level and grid resolution.
- Full statement of experimental results.
- Verification and validation tests performed by the author(s).
- Availability of computer code, input data and output data.
- Instructions for repeating computational experiments described in the paper.
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).
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.
"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].
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.”

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

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

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) was recently awarded the 2015 “Test of Time Award” from the ACM / IEEE Supercomputing Conference.

New ways to fool the masses in high-performance computing

- 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).
Numerical reproducibility

The ICERM reproducibility report further noted:

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.

The growing problem of numerical reliability

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.

Particularly vulnerable are:

1. Large-scale, highly parallel simulations, running on systems with hundreds of thousands or millions of processors — numerical sensitivities are greatly magnified.
2. Certain applications with highly ill-conditioned linear systems.
3. Large summations, especially those involving cancellations.
4. Long-time, iterative simulations (such as molecular dynamics or climate models).
5. Computations to resolve small-scale phenomena.
6. 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?
Have we learned anything?

Progress has been made, but a renewed focus is needed on:

- Supporting rigorous peer review.
- Focusing on end-to-end application run time.
- Reporting full details of computational environment, so results can be reproduced.
- Reporting all results — not selecting the best run out of many.
- Basing operation counts (for performance reporting) on the best practical serial algorithm.
- Reporting performance in a realistic operating environment, with no special settings inappropriate for common production usage.
- Being careful with numerical reproducibility. With the exploding size and scope of computations, numerical round-off error is much more severe. Have the authors fully justified the numerical reliability of their results?

THANKS!

This talk is available at: http://www.davidhbailey.com/dhbtalks/dhb-ipdps-2017.pdf