The NAS Parallel Benchmarks: History and Impact

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

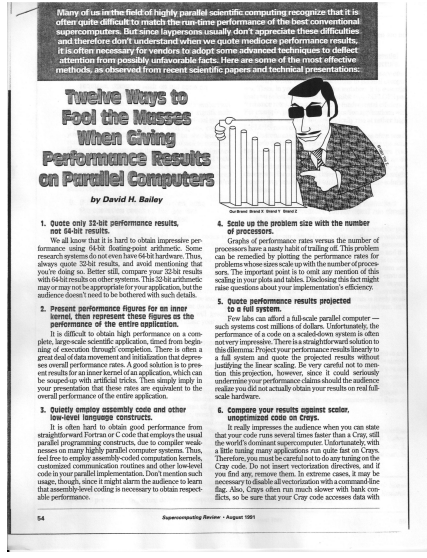
Co-authors: Eric Barszcz (NASA), John T. Barton, David Browning, Russell Carter (Esturion), Leo Dagum (ASSIA), Rod Fatoohi (San Jose State Univ.), Paul Frederickson (MathCube), Thomas Lasinski (LLNL, retired), Robert Schreiber (HP), Horst Simon (LBNL), Venkat Venkatakrishnan (CD-adapco), Sisira Weeratunga (LLNL)

November 11, 2015

Numerical Aerodynamic Simulation (NAS) Center staff photo, circa 1995

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 performers 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

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.

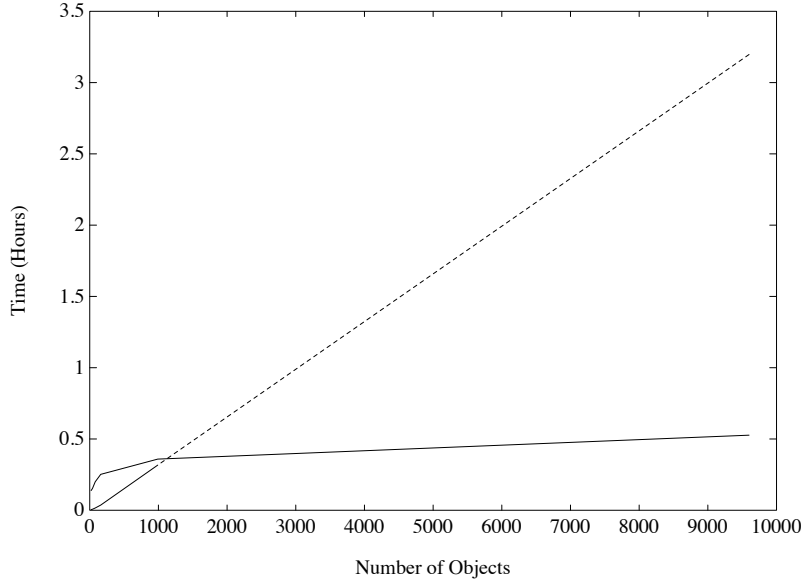
Typical excuse: Other algorithms are "more appropriate" for our parallel system.

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.

- 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.

The original NAS Parallel Benchmarks (NPB)

- The benchmarks were specified as a set of eight problems, defined in great detail in a technical document — a "paper and pencil design."
- The problem specifications included details such as the exact formulas to generate data (even pseudorandom data), what computations were to be timed, and validity tests of final results.
- A set of "reference implementations," targeted to three classes of architectures (vector, message passing, and SIMD) were included.
- Ground rules for submissions were given.
- The NPB originally specified Class A and Class B problem sizes; later Class C, D and E were added.
- Vendors and others were invited to submit results.

The eight original NPB

1. EP: An "embarrassingly parallel" problem, which evaluates an integral by means of pseudorandom trials.
2. MG: A simplified multigrid calculation.
3. CG: Uses a conjugate gradient method to compute the smallest eigenvalue of a large, sparse, symmetric positive definite matrix.
4. FT: Solves a 3-D partial differential equation system using FFTs.
5. IS: A large integer sort, typical of particle method codes.
6. LU: Solves a regular-sparse block lower-upper triangular system, typical of implicit computational fluid dynamics (CFD) calculations.
7. SP: Solves multiple systems of non-diagonally dominant, scalar pentadiagonal systems, typical of another class of implicit CFD.
8. BT: Solves multiple block tridiagonal systems, again typical of some implicit CFD calculations.

Some ground rules for NPB submissions

- All floating-point operations must be 64-bit.
- Code must be standard-conforming Fortran or C.
- Any language extension, compiler directive or library routine must be supported by the vendor for all users.
- Results must satisfy the tests provided for each benchmark.
- Allowable extensions, compiler directives and library routines:
  1. Constructs that specify allocation, communication or organization of data.
  2. Constructs that rearrange data.
  3. Constructs that synchronize actions between nodes.
  4. Constructs that perform array reduction operations.
  5. Constructs to perform a handful of well-known functions (e.g., matrix-matrix multiplication, FFTs, etc.).
  6. Constructs that perform high-speed I/O.

Early results from vendors

- Cray (Cray XMP, YMP and T3D/E).
- IBM RS/6000 parallel systems.
- Intel Touchstone.
- Thinking Machines CM2 and CM5.

Vendors continued to regularly submit results, and several supercomputer centers continued to specify the NPB for procurements, for several years.

What happened to the NPB?

- In 1997, NASA decided to cut back its research and development effort in high-performance computing.
- At or about this time, many of the NPB team left NASA Ames.
- Some development and support continued at NASA and at LBNL.

Current status:

- The NPB continues to be supported by researchers at NASA Ames Research Center.
- Several new benchmarks have been added to the suite.
- For details, contact Haoqiang Jin: haoqiang.jin@nasa.gov.

New benchmarks added to the NPB

Multi-zone versions of BT, SP and LU, designed to exploit multiple levels of parallelism in applications:

- BT-MZ: Uneven-size zones within a problem class, with an increasing number of zones as problem class grows.
- SP-MZ: Even-size zones within a problem class, with an increasing number of zones as problem class grows.
- LU-MZ: Even-size zones within a problem class, with a fixed number of zones for all problem classes.

Benchmarks for unstructured computation, parallel I/O, and data movement:

- UA: Unstructured Adaptive mesh, dynamic and irregular memory access.
- BT-IO: Test of different parallel I/O techniques.
- DC: Data Cube for data-intensive applications.
- DT: Data Traffic for data-intensive applications.

The NPB legacy: Computer science research

The NPB continue to be utilized and cited in computer science research:

- 3191 citations, according to Google Scholar (as of 4 November 2015).
- Studies include analyses of architectures, compilers, data movement, network design, programming languages and tools.

NPB reincarnated: The Berkeley "Motifs" and "TORCH" benchmarks

- In 2004, researchers at U.C. Berkeley devised a new set of performance benchmarks, known as the "Seven Motifs," later expanded to 13.
- In 2010, researchers at LBNL and U.C. Berkeley further developed these into the "TORCH" benchmarks.
- These benchmarks followed the NPB "pencil and paper" design, with problems defined in a technical document, accompanied with reference implementations.

References:

- K. Asanovic, R. Bodik, B. Catanzaro, J. Gebis, P. Husbands, K. Keutzer, D. Patterson, W. Plishker, J. Shalf, S. Williams and K. Yelick, "The landscape of parallel computing research: A view from Berkeley," Tech. Rep. UCB/EECS-2006-183, University of California, Berkeley, Dec. 2006, `http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.pdf`.
- E. Strohmaier, S. Williams, A. Kaiser, K. Madduri, K. Ibrahim, D. H. Bailey, J. W. Demmel, "A kernel testbed for parallel architecture, language, and performance research," *Proceedings of the International Conference of Numerical Analysis and Applied Mathematics* (ICNAAM), June 2010.

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 2015 "backtest overfitting" emerged as a major problem in computational finance.

Reproducibility Project staff

Credit: NY Times

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

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).

Recent case study: A GPU electronic structure code

A recent paper reported a GPU implementation of an electronic structure calculation, with a speedup over 600X. It was an interesting study, with solid results. However:

- GPU code utilized three GPUs, compared with only one core of a quad-core CPU.
- GPU calculations were in single precision, while CPU calculations were in double.
- CPU code employed a well-known computational chemistry library, which has good functionality but not state-of-the-art performance.
- GPU code completely changed the memory structure for better locality, but similar changes were not made to CPU code.
- In a separate study, researchers comparing the above GPU code to an efficient CPU code found speedups ranging up to 4.1, with an average of 2.2.

Conclusion: While this was a valuable study, it arguably hyped the GPU advantage.

Do all of us (once again!) need to be more careful in stating our results, particularly in abstracts and conclusions?

24 years of the NPB: What have we learned?

High-quality benchmarks are essential for real progress in HPC:

- Eliminating hype and confusion.
- Aiding intelligent system procurements.
- Providing a common platform for computer science research.

However:

- A good benchmark suite requires sustained organizational support.
- Benchmarks must be regularly upgraded in problem sizes, expanded to encompass new applications, and possibly even restructured when architectures are sufficiently changed.

Thanks! (on behalf of the entire NPB team)

This talk is available at:
http://www.davidhbailey.com/dhbtalks/dhb-npb-sc15.pdf