TORSTEN HOEFLER, ROBERTO BELLI

Scientific Benchmarking of Parallel Computing Systems
Twelve ways to tell the masses when reporting performance results

Performance Engineering for HPC Workshop at ICS’17, Frankfurt, Germany
Disclaimer(s)

- This is an experience talk (published at SC 15 – State of the Practice)!
  - Explained in SC15 FAQ:
    
    generalizable insights as gained from experiences with particular HPC machines/operations/applications/benchmarks, overall analysis of the status quo of a particular metric of the entire field or historical reviews of the progress of the field.
  - Don’t expect novel insights
    
    Given the papers I read, much of what I say may be new for many

- My musings shall not offend anybody
  - Everything is (now) anonymized

- Criticism may be rhetorically exaggerated
  - Watch for tropes!

- This talk should be entertaining!
High **Performance Computing**

dgemm("N", "N", 50, 50, 50, 1.0, A, 50, B, 50, 1.0, C, 50);

**Performance is nondeterministic and not modular**

Performance of complex systems is tricky
HPC is used to solve complex problems!

Treat performance-centric programming and system design like physical systems.
Scientific Performance Engineering

1) Observe
2) Model
3) Understand
4) Build
Part I: Observe

- Measure systems
- Collect data
- Examine documentation
- Gather statistics
- Document process
- Experimental design
- Factorial design
How does Garth measure and report performance?

- We may be interested in High Performance Computing
  - We (want to) see it as a science – reproducing experiments is a major pillar of the scientific method

- When measuring performance, important questions are
  - “How many iterations do I have to run per measurement?”
  - “How many measurements should I run?”
  - “Once I have all data, how do I summarize it into a single number?”
  - “How do I compare the performance of different systems?”
  - “How do I measure time in a parallel system?”
  - …

- How are they answered in the field today?
  - Let me start with a little anecdote … a reaction to this paper 😊
Original findings:

- If carefully tuned, NBC speed up a 3D solver

  Full code published

- 800^3 domain – 4 GB (distributed) array

  1 process per node, 8-96 nodes

  Opteron 246 (old even in 2006, retired now)

- Super-linear speedup for 96 nodes

  ~5% better than linear

9 years later: attempt to reproduce 😊!

System A: 28 quad-core nodes, Xeon E5520

System B: 4 nodes, dual Opteron 6274

“Neither the experiment in A nor the one in B could reproduce the results presented in the original paper, where the usage of the NBC library resulted in a performance gain for practically all node counts, reaching a superlinear speedup for 96 cores (explained as being due to cache effects in the inner part of the matrix vector product).”
State of the Practice in HPC

- Stratified random sample of three top-conferences over four years
  - 10 random papers from each (10-50% of population)
  - 120 total papers, 20% (25) did not report performance (were excluded)

Main results:
1. Most papers report details about the hardware but fail to describe the software environment.
   - Important details for reproducibility missing
2. The average paper’s results are hard to interpret and easy to question
   - Measurements and data not well explained
3. No statistically significant evidence for improvement over the years 😞

Our main thesis:
Performance results are often nearly impossible to reproduce! Thus, we need to provide enough information to allow scientists to understand the experiment, draw own conclusions, assess their certainty, and possibly generalize results.

This is especially important for HPC conferences and activities such as the Gordon Bell award!
Well, we all know this - but do we really know how to fix it?

1991 – the classic!
Twelve Ways to Fool the Masses When Giving Performance Results on Parallel Computers

2012 – the shocking
How did this happen?

2013 – the extension

Fooling the Masses with Performance Results: Old Classics & Some New Ideas

Gerhard Wellein\(^{(1,2)}\), Georg Hager\(^{(2)}\)

\(^{(1)}\)Department for Computer Science
\(^{(2)}\)Erlangen Regional Computing Center
Friedrich-Alexander-Universität Erlangen-Nürnberg
Our constructive approach: provide a set of (12) rules

- Attempt to emphasize interpretability of performance experiments
- The set is not complete
  - And probably never will be
  - Intended to serve as a solid start
  - Call to the community to extend it
- I will illustrate the 12 rules now
  - Using real-world examples
    All anonymized!
  - Garth and Eddie will represent the bad/good scientist
The most common issue: speedup plots

- Most common and oldest-known issue
  - First seen 1988 – also included in Bailey’s 12 ways
  - 39 papers reported speedups
    - 15 (38%) did not specify the base-performance 😞
  - Recently rediscovered in the “big data” universe
    - F. McSherry et al.: Scalability! but at what cost?, HotOS 2015
The most common issue: speedup plots

Check out my wonderful Speedup!

Rule 1: When publishing parallel speedup, report if the base case is a single parallel process or best serial execution, as well as the absolute execution performance of the base case.

- Most common and oldest known issue
  - First seen 1988 – also included in Bailey’s 12 ways
  - 39 papers reported speedups
  - 15 (38%) did not specify the base-performance

- Recently rediscovered in the “big data” universe
  - F. McSherry et al.: Scalability! but at what cost?, HotOS 2015

I can’t tell if this is useful at all!
Rule 2: Specify the reason for only reporting subsets of standard benchmarks or applications or not using all system resources.

- This implies: Show results even if your code/approach stops scaling!
Rule 3: *Use the arithmetic mean only for summarizing costs. Use the harmonic mean for summarizing rates.*

Rule 4: *Avoid summarizing ratios; summarize the costs or rates that the ratios base on instead. Only if these are not available use the geometric mean for summarizing ratios.*

- 51 papers use means to summarize data, only four (!) specify which mean was used
- A single paper correctly specifies the use of the harmonic mean
- Two use geometric means, without reason
- Similar issues in other communities (PLDI, CGO, LCTES) – see N. Amaral’s report
- Harmonic mean ≤ geometric mean ≤ arithmetic mean

The latency of Piz Dora is 1.77us! How did you get to this? I averaged 10 tests, it must be right!

**Rule 5:** Report if the measurement values are deterministic. For nondeterministic data, report confidence intervals of the measurement.

- Most papers report nondeterministic measurement results
  - Only 15 mention some measure of variance
  - Only two (!) report confidence intervals

- CIs allow us to compute the number of required measurements!

- Can be very simple, e.g., single sentence in evaluation:
  “We collected measurements until the 99% confidence interval was within 5% of our reported means.”
Dealing with variation

The confidence interval is 1.765us to 1.775us.

Did you assume normality?

Yes, I used the central limit theorem to normalize by summing subsets of size 100!

Can we test for normality?

No, the data is not normal at all! The real CI is actually 1.6us to 1.9us!

Rule 6: Do not assume normality of collected data (e.g., based on the number of samples) without diagnostic checking.

- Most events will slow down performance
  - Heavy right-tailed distributions

- The Central Limit Theorem only applies asymptotically
  - Some papers/textbook mention “30-40 samples”, don’t trust them!

- Two papers used CIs around the mean without testing for normality
Dealing with non-normal data – nonparametric statistics

- Rank-based measures (no assumption about distribution)
  - Essentially always better than assuming normality
- Example: median (50th percentile) vs. mean for HPL
  - Rather stable statistic for expectation
  - Other percentiles (usually 25th and 75th) are also useful
Comparing nondeterministic measurements

I saw variance using GarthCC as well!

Show me the data!

Retract the paper! You have not shown anything!
What if the data looks weird!?

Look what data I got!

Clearly, the mean/median are not sufficient!

Try quantile regression!
Rule 8: Carefully investigate if measures of central tendency such as mean or median are useful to report. Some problems, such as worst-case latency, may require other percentiles.

How many measurements are needed?

- Measurements can be expensive!
  - Yet necessary to reach certain confidence

- How to determine the minimal number of measurements?
  - Measure until the confidence interval has a certain acceptable width
  - For example, measure until the 95% CI is within 5% of the mean/median
  - Can be computed analytically assuming normal data
  - Compute iteratively for nonparametric statistics

- Often heard: “we cannot afford more than a single measurement”
  - E.g., Gordon Bell runs
  - Well, then one cannot say anything about the variance
    - Even 3-4 measurement can provide very tight CI (assuming normality)
    - Can also exploit repetitive nature of many applications

Rule 9: Document all varying factors and their levels as well as the complete experimental setup (e.g., software, hardware, techniques) to facilitate reproducibility and provide interpretability.

- We recommend factorial design
- Consider parameters such as node allocation, process-to-node mapping, network or node contention
  - If they cannot be controlled easily, use randomization and model them as random variable
- This is hard in practice and not easy to capture in rules
Time in parallel systems

My simple broadcast takes only one latency!

That's nonsense!

But I measured it so it must be true!

Measure each operation separately!

t = MPI_Wtime();
for(i=0; i<1000; i++) {
    MPI_Bcast(...);
}
t += MPI_Wtime();
t /= 1000;
Rule 10: *For parallel time measurements, report all measurement, (optional) synchronization, and summarization techniques.*

- Measure events separately
  - Use high-precision timers
  - Synchronize processes

- Summarize across processes:
  - Min/max (unstable), average, median – depends on use-case

---

Give times a meaning!

I compute $10^{10}$ digits of Pi in 2ms on Dora!

I have no clue.

Can you provide?
- Ideal speedup
- Amdahl’s speedup
- Parallel overheads

Ok: The code runs 17ms on a single core, 0.2ms are initialization, which has collapsed.

Rule 11: If possible, show upper performance bounds to facilitate interpretability of the measured results.

- Model computer system as k-dimensional space
  - Each dimension represents a capability
    - Floating point, Integer, memory bandwidth, cache bandwidth, etc.
  - Features are typical rates
  - Determine maximum rate for each dimension
    - E.g., from documentation or benchmarks
  - Can be used to proof optimality of implementation
    - If the requirements of the bottleneck dimension are minimal

Rule 12: Plot as much information as needed to interpret the experimental results. Only connect measurements by lines if they indicate trends and the interpolation is valid.
We have the (statistically sound) data, now what?

The 99% confidence interval is within 1% of the reported median.

Matrix Multiply

\[ t(n) = a \cdot n^3 \]

We have the (statistically sound) data, now what?

The 99% confidence interval is within 1% of the reported median.

The adjusted $R^2$ of the model fit is 0.99

Performance Modeling = Performance Analysis v 2.0 (the next logical step)
Part II: Model

Burnham, Anderson: “A model is a simplification or approximation of reality and hence will not reflect all of reality. ... Box noted that “all models are wrong, but some are useful.” While a model can never be “truth,” a model might be ranked from very useful, to useful, to somewhat useful to, finally, essentially useless.”

This is generally true for all kinds of modeling. We focus on performance modeling in the following!
Performance Modeling

Capability Model

Performance Model

Requirements Model

Application Expertise

Systems Expertise
Requirements modeling I: Six-step performance modeling

1. Input parameters
2. Describe application kernels
3. Communication pattern
4. Communication / computation overlap
5. Fit sequential baseline
6. Communication parameters

10-20% speedup [2]

Requirements modeling II: Automated best-fit modeling

- Manual kernel selection and hypothesis generation is time consuming (boring and tricky)
- Idea: Automatically select best (scalability) model from predefined search space

\[ f(p) = \sum_{k=1}^{n} c_k \cdot p^{i_k} \cdot \log_{2}^{j_k} (p) \]

- \( n = 1 \)
- \( I = \{0, 1, 2\} \)
- \( J = \{0, 1\} \)

\[ n \in \mathbb{N} \quad i_k \in I \quad j_k \in J \quad I, J \subseteq \mathbb{Q} \]

- Number of processes
- (model) constant
- Number of terms

\[ c_1 \times \log(p) \]
\[ c_1 \times p \times \log(p) \]
\[ c_1 \times p^2 \times \log(p) \]

Requirements modeling II: Automated best-fit modeling

- Manual kernel selection and hypothesis generation is time consuming (and boring)
- Idea: Automatically select best model from predefined space

\[ f(p) = \sum_{k=1}^{n} c_k \times p^{i_k} \times \log^{j_k}(p) \]

- \( n = 2 \)
- \( I = \{0, 1, 2\} \)
- \( J = \{0, 1\} \)

\[ c_1 \cdot \log(p) + c_2 \cdot p \]
\[ c_1 \cdot \log(p) + c_2 \cdot p \cdot \log(p) \]
\[ c_1 \cdot \log(p) + c_2 \cdot p^2 \]
\[ c_1 \cdot p + c_2 \cdot p \cdot \log(p) \]
\[ c_1 \cdot p + c_2 \cdot p^2 \]
\[ c_1 \cdot p + c_2 \cdot p^2 \cdot \log(p) \]
\[ c_1 \cdot p \cdot \log(p) + c_2 \cdot p^2 \]
\[ c_1 \cdot p \cdot \log(p) + c_2 \cdot p^2 \cdot \log(p) \]
\[ c_1 \cdot p^2 + c_2 \cdot p^2 \cdot \log(p) \]

Tool support: Extra-P for automated best-fit modeling [1]


Requirements modeling III: Source-code analysis [1]

- Extra-P selects model based on best fit to the data
  - What if the data is not sufficient or too noisy?
- Back to first principles
  - The source code describes all possible executions
  - Describing all possibilities is too expensive, focus on counting loop iterations symbolically

\[
N = (n+1) \log_2 n - n + 2
\]

\[
W = N \bigg|_{p=1}
\]

\[
D = N \bigg|_{p \to \infty}
\]

[1]: TH, G. Kwasniewski: Automatic Complexity Analysis of Explicitly Parallel Programs, ACM SPAA’14
TH: Bridging Performance Analysis Tools and Analytic Performance Modeling for HPC
Capability models for network communication

The LogP model family and the LogGOPS model [1]

Finding LogGOPS parameters

Netgauge [2], model from first principles, fit to data using special kernels

Large scale LogGOPS Simulation

LogGOPSim [1], simulates LogGOPS with 10 million MPI ranks

<5% error

2) Design optimal algorithms – small broadcast in LogP

L=2, o=1, P=7

Capability models for cache-to-cache communication


Local read: \( R_L \approx 3.8 \text{ ns} \)
Remote read: \( R_R \approx 115 \text{ ns} \)
Invalid read: \( R_I \approx 135 \text{ ns} \)
Model-tuned Barrier and Reduce vs. Intel’s OpenMP and MPI

(a) Filling Tiles.  
(b) Scatter.

(a) Filling Tiles.  
(b) Scatter.

Barrier (7x faster than OpenMP)  
Reduce (5x faster than OpenMP)
Performance Model
Conclusions and call for action

- **Performance may not be reproducible**
  - At least not for many (important) results

- **Interpretability fosters scientific progress**
  - Enables to build on results
  - Sounds statistics is the biggest gap today

- **We need to foster interpretability**
  - Do it ourselves (this is not easy)
  - Teach young students
  - Maybe even enforce in TPCs

- **See the 12 rules as a start**
  - Need to be extended (or concretized)
  - Much is implemented in LibSciBench [1]

[1]: http://spcl.inf.ethz.ch/Research/Performance/LibLSB/

Acknowledgments

- ETH’s mathematics department (home of R)
  - Hans Rudolf Künsch, Martin Maechler, and Robert Gantner

- Comments on early drafts
  - David H. Bailey, William T. Kramer, Matthias Hauswirth, Timothy Roscoe, Gustavo Alonso, Georg Hager, Jesper Träff, and Sascha Hunold

- Help with HPL run
  - Gilles Fourestier (CSCS) and Massimiliano Fatica (NVIDIA)