Performance Benchmarking

High Performance Computing, Summer 2021

Biagio Cosenza
Department of Computer Science
University of Salerno
bcosenza@unisa.it

Twelve Ways to Fool the Masses When Giving Performance Results on Parallel Computers
David H. Bailey
June 11, 1991
Ref: Supercomputing Review, Aug. 1991, pg. 54--55

Abstract

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. This humorous article outlines twelve ways commonly used in scientific papers and presentations to artificially boost performance rates and to present these results in the "best possible light" compared to other systems.

The author is with the Numerical Aerodynamic Sim
NASA Ames Research Center, Moffett Field, CA

Many of us in the field of highly parallel scientific
quite difficult to match the run time performance o
supercomputers. But since lay persons usually don
therefore don't understand when we quote mediocr
necessary for us to adopt some advanced technique
possibly unfavorable facts. Here are some of the m
from recent scientific papers and technical presenta

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

We all know that it is hard to obtain impressive per
arithmetic. Some research systems do not even hav
32-bit results, and avoid mentioning this fact if at a
32-bit results with 64-bit results on other systems.
appropriate for your application, but the audience d
details.

2. Present performance figures for an inner kernel,
performance of the entire application.

It is quite difficult to obtain high performance on a
application, timed from beginning of execution thr
great deal of data movement and initialization that
A good solution to this dilemma is to present result
which can be souped up with artificial tricks. Then
rates are equivalent to the overall performance of t

3. Quietly employ assembly code and other low-le

Fooling the masses with performance results:
Old classics and some new ideas

G. Wellein and G. Hager
Department for Computer Science and Erlangen Regional Computing Center
Friedrich-Alexander-Universität Erlangen-Nürnberg

In 1991, David H. Bailey published his insightful "Twelve Ways to Fool the Masses When Giving Performance Results on Parallel Computers." In that humorous article, Bailey pinpointed typical "evade and disguise" techniques for presenting mediocre performance results in the best possible light. At that time, the supercomputing landscape was governed by the "chicken vs. oxen" debate: Could strong vector CPUs survive against the new massively parallel systems? In the past two decades, hybrid, hierarchical systems, multi-core processors, accelerator technology, and the dominating presence of commodity hardware have reshaped the landscape of High Performance Computing. It's also not so much oxen vs. chickens anymore; billions of ants have entered the battlefield. This talk gives an update of the "Twelve Ways." Old classics are presented alongside new "stunts" that reflect today's technological boundary conditions.

DISCLAIMER: Although these musings are certainly inspired by experience with many publications and talks in HPC, I wish to point out that (i) no offense is intended, (ii) I am not immune to the inherent temptations myself and (iii) this all still just meant to be fun.

How did *this* get published?
Pitfalls in experimental evaluation of computing systems

José Nelson Amaral
University of Alberta
Edmonton, AB, Canada

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

Torsten Hoefler
Dept. of Computer Science
ETH Zurich
Zurich, Switzerland
htor@inf.ethz.ch

Roberto Belli
Dept. of Computer Science
ETH Zurich
Zurich, Switzerland
bellir@inf.ethz.ch

ABSTRACT

Measuring and reporting performance of parallel computers constitutes the basis for scientific advancement of high-performance computing (HPC). Most scientific reports show performance improvements of new techniques and are thus obliged to ensure reproducibility or at least interpretability. Our investigation of a stratified sample of 120 papers across three top conferences in the field shows that the state of the practice is lacking. For example, it is often unclear if reported improvements are deterministic or observed by chance. In addition to distilling best practices from existing work, we propose statistically sound analysis and reporting techniques and simple guidelines for experimental design in parallel computing and codify them in a portable benchmarking library. We aim to improve the standards of reporting research results and initiate a discussion in the HPC field. A wide adoption of our minimal set of rules will lead to better interpretability of performance results and improve the scientific culture in HPC.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

Keywords

Benchmarking, parallel computing, statistics, data analysis

1. INTRODUCTION

Correctly designing insightful experiments to measure and report performance numbers is a challenging task. Yet, there is surprisingly little agreement on standard techniques for measuring, reporting, and interpreting computer performance. For example, common questions such as "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?", or "How do I measure time in a parallel system?" are usually answered based on intuition. While we believe that an expert's intuition is most often correct, there are cases where it fails and invalidates expensive experiments or even misleads us. Bailey [3] illustrates this in several common but misleading data reporting patterns that he and his colleagues have observed in practice.

SC '15, November 15 - 20, 2015, Austin, TX, USA

ISBN 978-1-4503-3723-6/15/11 ... $15.00
DOI: http://dx.doi.org/10.1145/2807591.2807644

Reproducing experiments is one of the main principles of the scientific method. It is well known that the performance of a computer program depends on the application, the input, the compiler, the runtime environment, the machine, and the measurement methodology [20,43]. If a single one of these aspects of *experimental design* is not appropriately motivated and described, presented results can hardly be reproduced and may even be misleading or incorrect.

The complexity and uniqueness of many supercomputers makes reproducibility a hard task. For example, it is practically impossible to recreate most hero-runs that utilize the world's largest machines because these machines are often unique and their software configurations changes regularly. We introduce the notion of *interpretability*, which is weaker than reproducibility. *We call an experiment interpretable if it provides enough information to allow scientists to understand the experiment, draw own conclusions, assess their certainty, and possibly generalize results.* In other words, interpretable experiments support sound conclusions and convey precise information among scientists. Obviously, every scientific paper should be interpretable; unfortunately, many are not.

For example, reporting that an High-Performance Linpack (HPL) run on 64 nodes (N=314k) of the Piz Daint system during normal operation (cf. Section 4.1.2) achieved 77.38 Tflop/s is hard to interpret. If we add that the theoretical peak is 94.5 Tflop/s, it becomes clearer, the benchmark achieves 81.8% of peak performance. But is this true for every run or a typical run? Figure 1

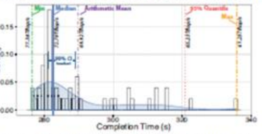


Figure 1: Distribution of completion times for 50 HPL runs.

provides a much more interpretable and informative representation of the collected runtimes of 50 executions. It shows that the variation is up to 20% and the slowest run was only 61.2 Tflop/s.

Our HPL example demonstrates that one of the most important aspects of ensuring interpretability is the sound analysis of the measurement data. Furthermore, the hardness of reproducing experiments makes an informative presentation of the collected data essential, especially if the performance results were nondeterministic. Potential sources of nondeterminism, or *noise*, can be the system (e.g., network background traffic, task scheduling, interrupts, job placement in the batch system), the application (e.g., load bal-

Outline

- Twelve ways to fool the masses when giving performance results on parallel computers
- Fooling the masses with performance results: Old classics and some new ideas
- How did this get published? Pitfalls in experimental evaluation of computing systems
- Twelve ways to tell the masses when reporting performance results

Twelve ways to fool the masses
when giving performance results on parallel computers

1. Quote only 32-bit performance results, not 64-bit results.
2. Present performance figures for an inner kernel, and then represent these figures as the performance of the entire application.
3. Quietly employ assembly code and other low-level language constructs.
4. Scale up the problem size with the number of processors, but omit any mention of this fact.
5. Quote performance results projected to a full system.
6. Compare your results against scalar, unoptimized code on Crays.

Twelve Ways to Fool the Masses When Giving Performance Results on Parallel Computers
David H. Bailey. June 11, 1991. Ref: Supercomputing Review, Aug. 1991, pg. 54--55

Twelve ways to fool the masses when giving performance results on parallel computers

7. When direct run time comparisons are required, compare with an old code on an obsolete system.
8. If MFLOPS rates must be quoted, base the operation count on the parallel implementation, not on the best sequential implementation.
9. Quote performance in terms of processor utilization, parallel speedups or MFLOPS per dollar.
10. Mutilate the algorithm used in the parallel implementation to match the architecture.
11. Measure parallel run times on a dedicated system, but measure conventional run times in a busy environment.
12. If all else fails, show pretty pictures and animated videos, and don't talk about performance.

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

David H. Bailey. June 11, 1991. Ref: Supercomputing Review, Aug. 1991, pg. 54--55

Fooling the masses with performance results: Old classics and some new ideas

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

Fooling the Masses with Performance Results: Old Classics & Some New Ideas. Jee Choi. https://blogs.fau.de/hager/archives/7312

- The problem with averages
  - reports on the wrong use of arithmetic average for aggregate normalized numbers

Scientific Benchmarking of Parallel Computing Systems

- 12 rules
- Attempt to emphasize interpretability of performance experiments
- Using real word examples

Torsten Hoefler, Roberto Belli
Scientific benchmarking of parallel computing systems: twelve ways to tell the masses when reporting performance results.
SC 2015: 73:1-73:12

- Motivation example
- LINPACK performance on 50 runs during normal operations
- Insights
  - Is the arithmetic mean representative of the performance?
  - What about median?
  - Distribution vs single metric

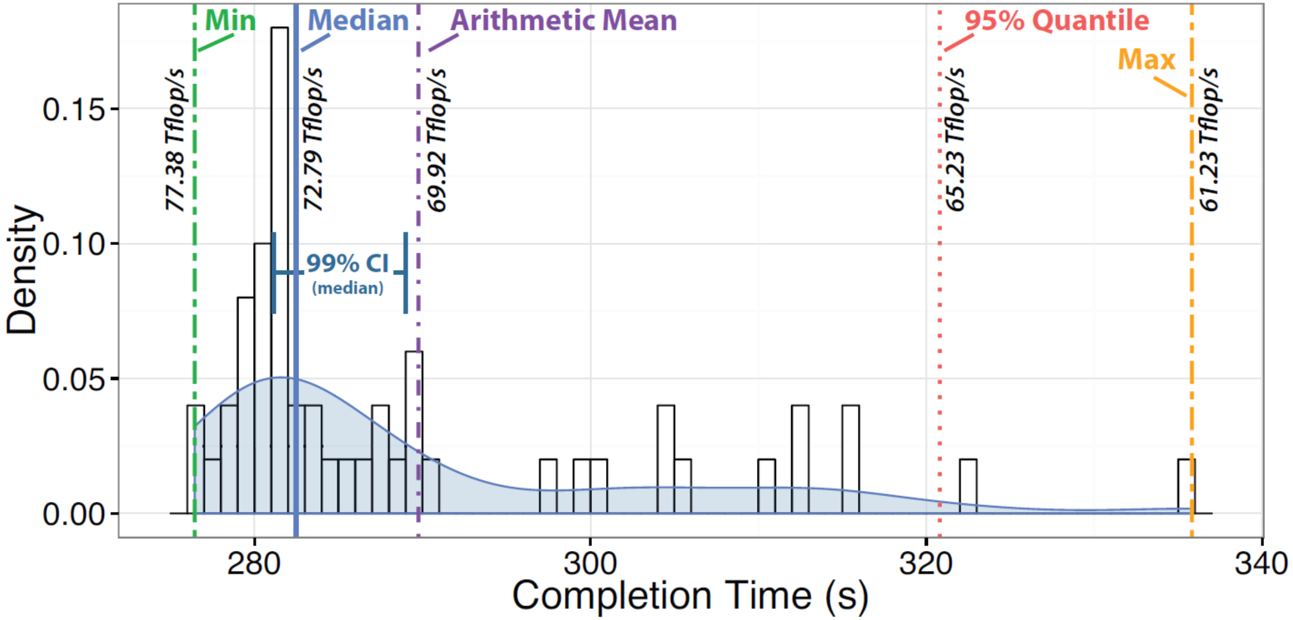


Figure 1: Distribution of completion times for 50 HPL runs.

Rule 1: Clarify Speedup Baseline

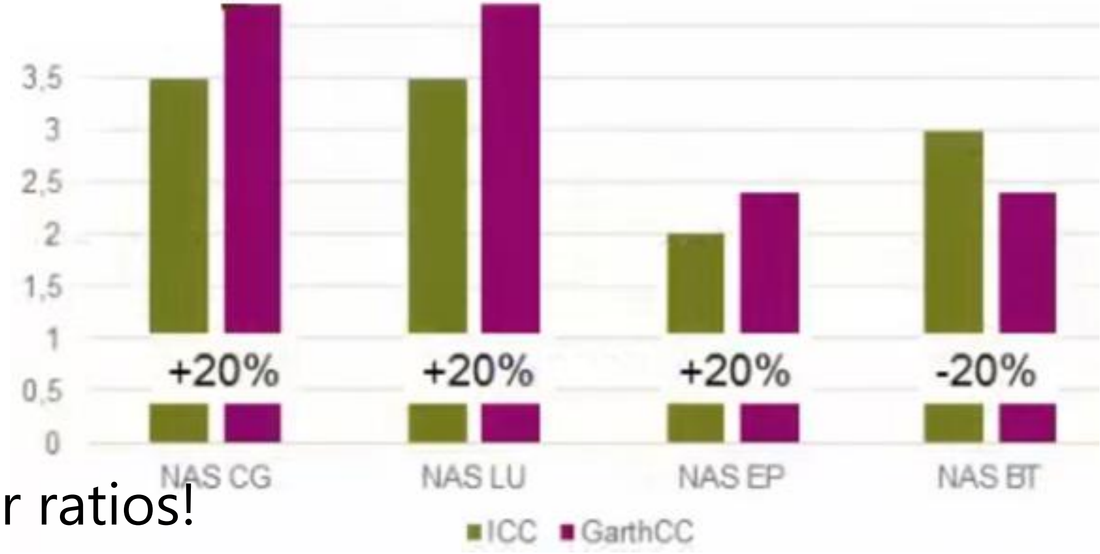
- Use Speedup with Care
- #1 When publishing parallel speedup, report if the base case is a single parallel process or best serial execution, as well as the absolute execution performance of the base case.
  - a simple generalization of this rule implies that one should never report ratios without absolute values
- Report Units Unambiguously
- Typical error in scalability plots

Rule 2: Benchmarks

#2 Specify the reason for only reporting subsets of standard benchmarks or applications or not using all system resources.

- Do not Cherry-pick
  - use the whole node
  - use the whole benchmark/application
- This implies: Show results even if your code/approach stops scaling!
- In general, one should compare to standard benchmarks or other papers where possible to increase interpretability.
  - a corollary of this rule is to report all results, not just the best.

Summarizing Results



- Example: Results of a GarthCC compiler
  - results on 4 benchmarks
- What are the performance?

"GarthCC is 10% faster than ICC in average"

- Problem: you cannot use arithmetic mean for ratios!

"GarthCC has 8% speedup on geometric mean"

- Problem: geometric mean has no clear interpretation
- What about complete performance?

"As BT takes longer, if we take actual runtimes GarthCC is 10% slower!"

Rule 3 and 4: Summarizing Results

- Summarizing costs (e.g., execution time): in the standard case where all measurements are weighted equally use the arithmetic mean to summarize costs

- Summarizing rates (flop/s, flop/watt, flop/inst): if the denominator has the primary semantic meaning, the harmonic mean provides correct results

#3 Use the arithmetic mean only for summarizing costs. Use the harmonic mean for summarizing rates.

- Summarizing ratios: (costs and rates have units, ratios do not)
  - ratios should never be averaged as such an average is meaningless
  - if a summary is needed then compute the correct mean of the costs or rates of the measurement before the normalization
  - if in exceptional situations, e.g, the cost measures are not available, normalized results have to be averaged, then they should be averaged using the geometric mean

#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.

- Harmonic mean ≤ geometric mean ≤ arithmetic mean

Rule 5: Reporting Non deterministic Results

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

- Experimental results are nondeterministic by nature
- Must report the statistical method to use.
  - standard deviation, coefficient of variation, confidence intervals of the mean
- If the data is nearly deterministic, then the reporting can be summarized, for example:
  - *We collected measurements until the 99% confidence interval was within 5% of our reported means.*

Rule 6

- It is common to assume that measurements are normally distributed
  - E.g., using the Central Limit Theorem (CLT), we affirm: “*the confidence interval for this metric is 1.76 sec to 1.78 sec*”
  - CLT: when independent random variables are added, their properly normalized sum tends toward a normal distribution (informally a bell curve)

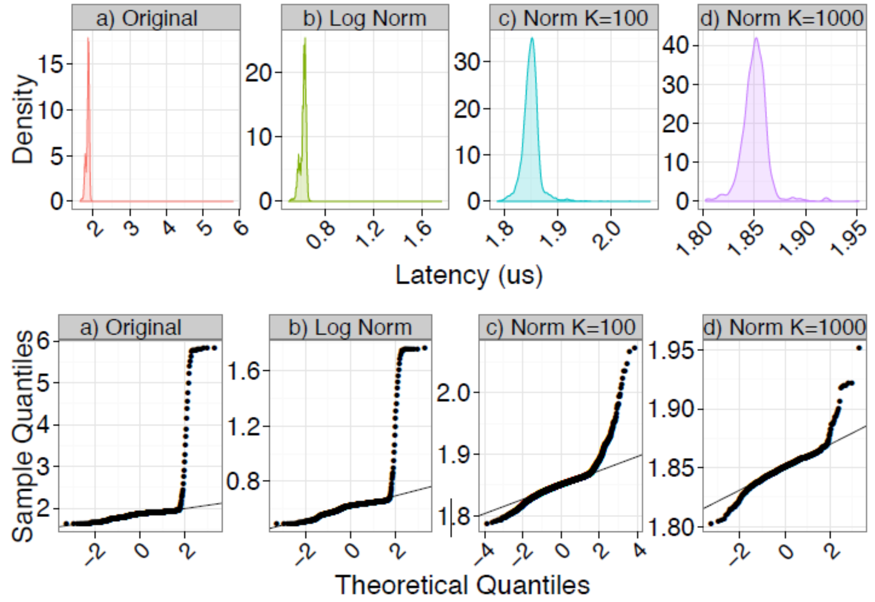


Figure 2: Normalization of 1M ping-pong samples on Piz Dora.

- Unfortunately, performance are not normal distr.
  - Heavy right-tailed distributions

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

Nonparametric statistics

- Rank-based measures are better
  - no assumption about distribution
- E.g.. mean vs median for the HPL benchmarks
  - Median is better
- Also, percentiles (25 and 75) are useful

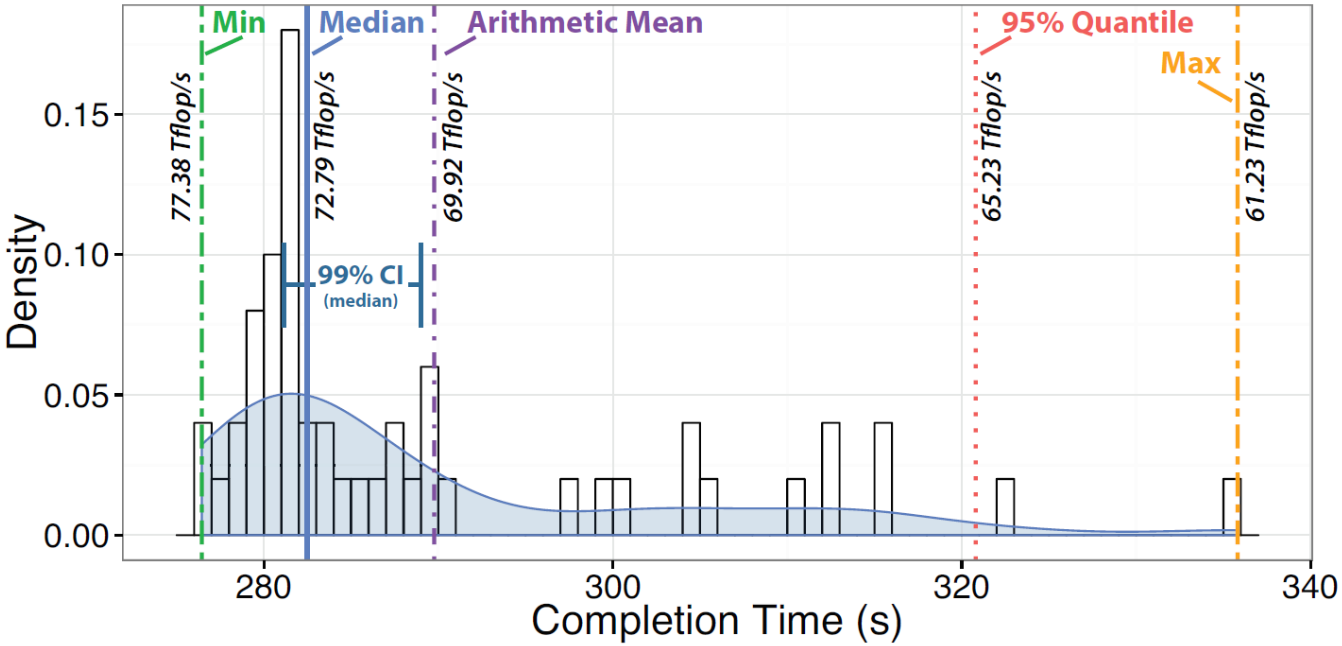
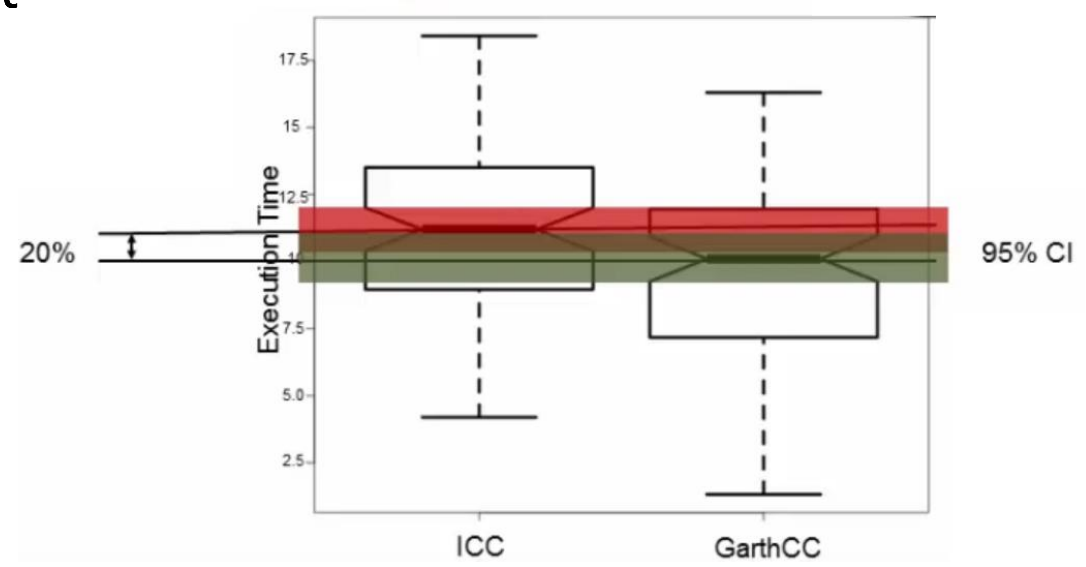


Figure 1: Distribution of completion times for 50 HPL runs.

Comparing Nondeterministic measurements

- If the interval of confidence overlaps, your data are not statistically significant
- E.g: this 20% improvement is not significant



- 7 Compare nondeterministic data in a statistically sound way, e.g., using non-overlapping confidence intervals or ANOVA.

Weird Distributions

- Example from a latency test on a Cry machine Piz Dora
- Clearly mean/median are not sufficient
  - Try quantile regression

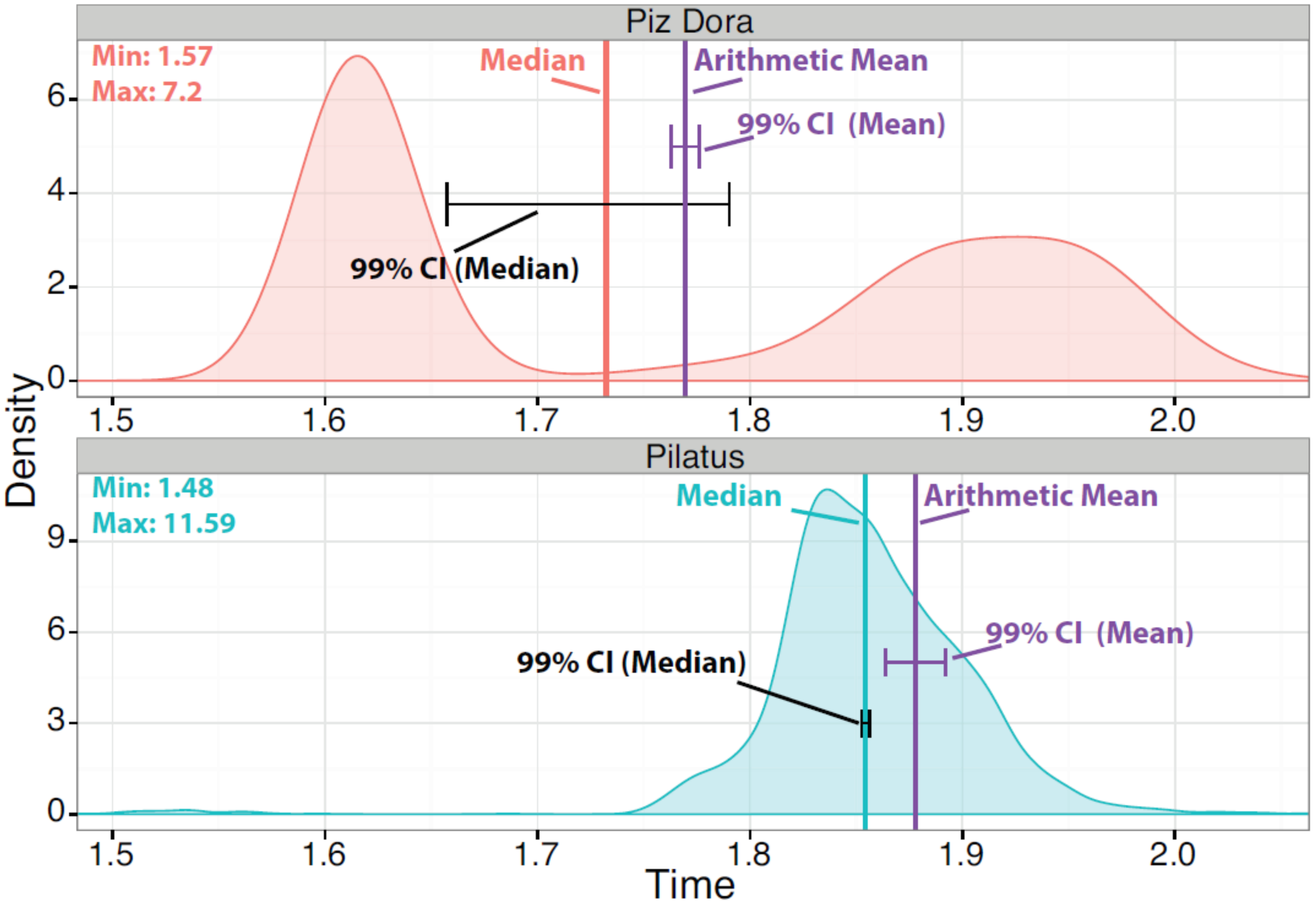


Figure 3: Significance of latency results on two systems.

Solution: Quantile Regression

- Quantile Regression
  - nonparametric measure
  - compare the effect across various ranks and is thus most useful if the effect appears at a certain percentile
- In this example:
  - Pilatus is better for worst-case latency-critical workloads
  - Dora is faster in the average case

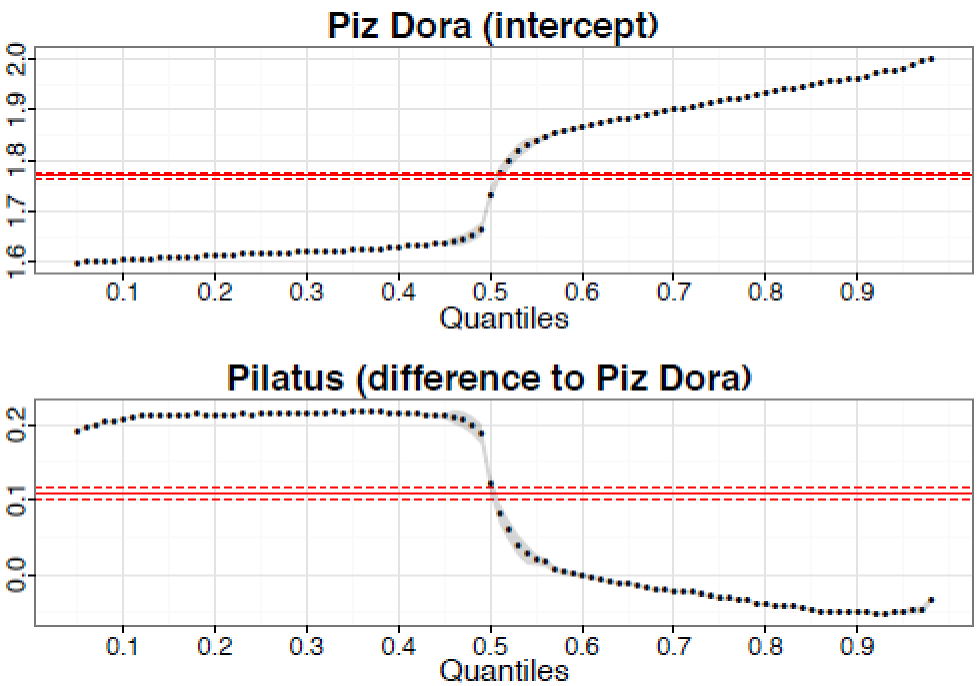


Figure 4: Quantile regression comparison of the latencies comparing Pilatus (base case or intercept) with Piz Dora.

Rule 8

#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.

- Consider quantile regression.

Augusto Born de Oliveira, Sebastian Fischmeister, Amer Diwan, Matthias Hauswirth, Peter F. Sweeney: Why you should care about quantile regression. ASPLOS 2013: 207-218

Rule 9

#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.

- Example: use power of two for the number of processors
  - But sometime algorithms are different for non-power-of-two number of processors
- Report on
  - Node allocation, process-to-node mapping, network, node contention, etc.
  - If they cannot be easily controlled, use randomization and model them as random variable

Rule 10

#10 For parallel time measurements, report all measurement, (optional) synchronization, and summarization techniques.

- Example: parallel runtime in MPI
  - most of today's parallel systems are asynchronous and do not have a common clock source
  - furthermore, clock drift between processes could impact measurements and network latency variations make time synchronization tricky
  - many evaluations use an (MPI or OpenMP) barrier to synchronize processes for time measurement.
    - unreliable because neither MPI nor OpenMP provides timing guarantees for their barrier calls
  - measuring single-processor time is also problematic

```
t = -MPI_Wtime();
for(i=0; i<1000; i++) {
  MPI_Bcast(...);
}
t += MPI_Wtime();
t /= 1000;
```

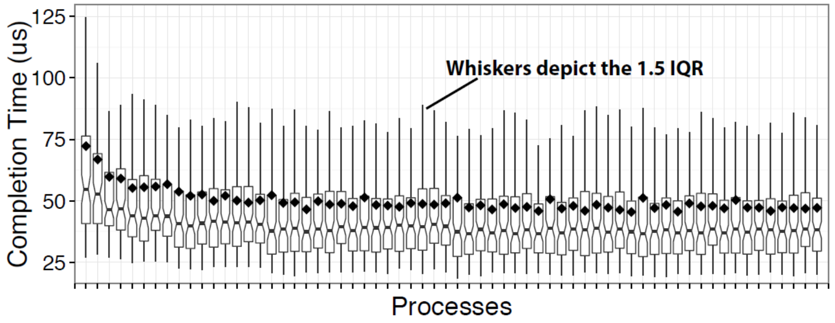
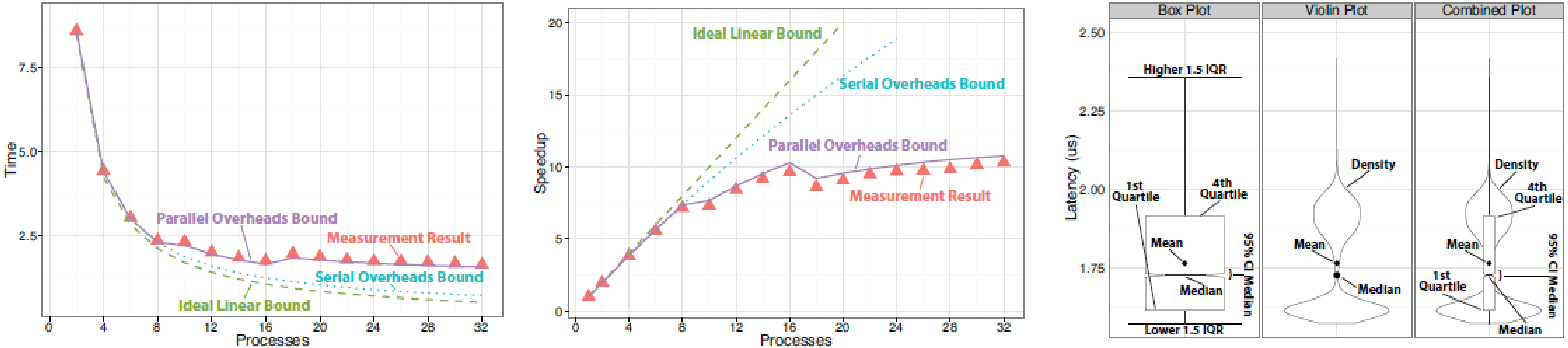


Figure 6: Variation across 64 processes in MPI_Reduce.

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

- Bounds modeling: ideal linear speedup, serial overheads (Amdahl's law), parallel overheads



(a) Time (b) Speedup (c) Box and Violin Plots

Figure 7: Time and speedup bounds models for parallel scaling and different plot types. Experiments for (a) and (b) were repeated ten times each and the 95% CI was within 5% of the mean. Plot (c) shows the latency of 10^6 64B ping-pong experiments on Piz Dora.

Rule 12

#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.

- Useful graphics: box plots, violin plot
- Useful to plot as much as information as possible

ACM Artifact Evaluation

- Repeatability (Same team, same experimental setup)
  - The measurement can be obtained with stated precision by the same team using the same measurement procedure, the same measuring system, under the same operating conditions, in the same location on multiple trials.
  - For computational experiments, this means that a researcher can reliably repeat her own computation.
- Reproducibility (Different team, same experimental setup)
  - The measurement can be obtained with stated precision by a different team using the same measurement procedure, the same measuring system, under the same operating conditions, in the same or a different location on multiple trials.
  - For computational experiments, this means that an independent group can obtain the same result using the author's own artifacts.
- Replicability (Different team, different experimental setup)
  - The measurement can be obtained with stated precision by a different team, a different measuring system, in a different location on multiple trials.
  - For computational experiments, this means that an independent group can obtain the same result using artifacts which they develop completely independently.

ACM Artifact Evaluation Badges

1. Artifact evaluated - Functional
2. Artifact evaluated – Reusable
3. Artifact available
4. Results reproduced
5. Results replicated

Project Report Notes

- Find at least 5 publications related to your work
  - In conference: SC, ICS, PPoPP, HPDC, IPDPS, PLDI, ICPP, EuroPar, …
  - In journal: TPDS, FGCS, …
- What is your contribution?
  - Write your report around the “contribution points”
    1. Introduction
    2. Background / Related Work
    3. Methodology
    4. Implementation
    5. Discussion
    6. Conclusion
  - Note: you can adapt the structure to your project
- Don’t be verbose!

HPC Lab

Please join your project Team Channel to work at your assigned project and talk with your assigned teaching assistant.