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

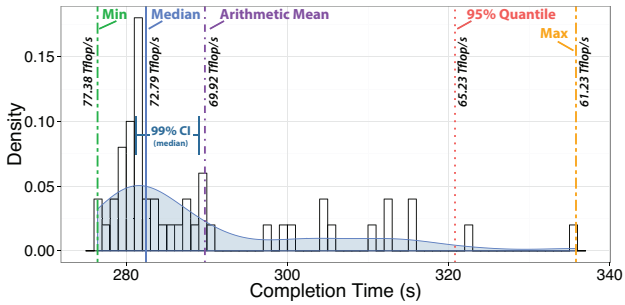


Figure 1: Distribution of completion times for 50 HPL runs.

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-

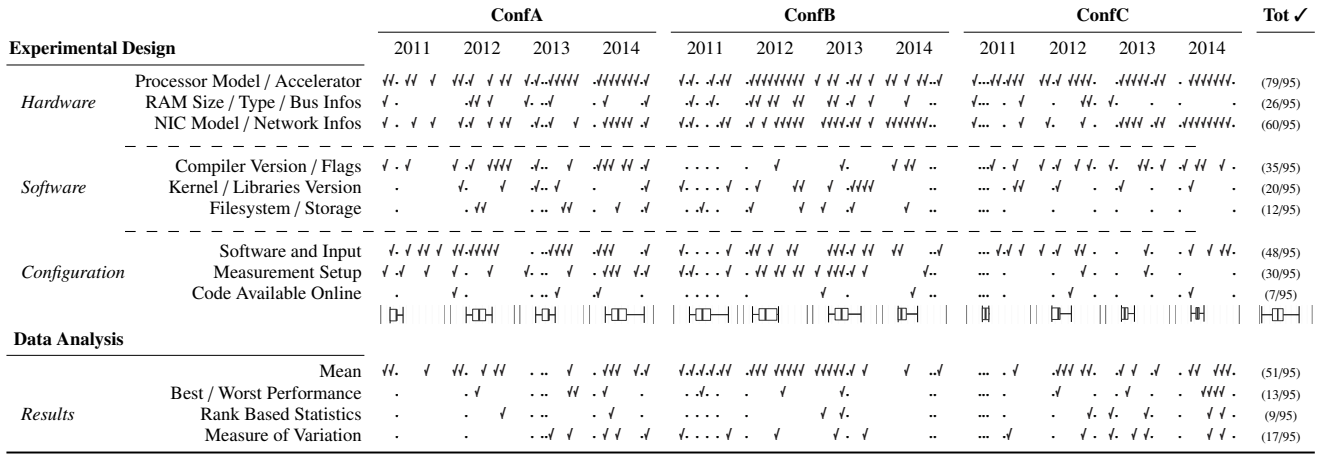


Table 1: Summary of the results of our literature survey. We report all data to support readers in drawing their own conclusions[1]; dots mark not applicable papers. Experimental design results are summarized for all 10 papers of each year in a horizontal box plot showing the distribution of the numbers of ✓ per paper ranging from 0 (left) to 9 (right); whiskers mark the min/max range.

ancing, cache misses, managed memory, JIT compilation), or the environment (e.g., other applications causing interference, or even the application's stack layout [43]). On many systems, such noise is small and its influence can be measured using simple statistical techniques but sometimes, noise can cause significant degradation of program execution [26,47]. Yet, most research publications employ simplistic approaches to report nondeterministic results.

We show in a literature review study that most papers from a representative sample of three leading conferences only use simple summarized performance metrics without specifying variation; many papers do not even mention the exact method how the data was summarized and are thus neither reproducible nor interpretable. In an attempt to improve the state of the practice, we propose simple ground rules and associated statistical techniques to design experiments, measure and summarize data, and report results in parallel computing and related disciplines. Some of these rules go beyond the state of the art while others are obvious to some readers. Yet, we consider it useful to explicitly state and collect them into a single document that can be used to define some qualitative level of interpretability. Our guidelines are intended to assist researchers, reviewers, and readers in designing, conducting, and evaluating empirical studies. Editorial boards and program committees may use this as a basis for developing guidelines for reviewers to establish policies for empirical studies.

2. STATE OF THE PRACTICE

We start our analysis by assessing the state of the practice in HPC research using a stratified random sample from the proceedings of the leading conferences ACM HPDC, ACM/IEEE Supercomputing (SC), and ACM PPoPP of the years 2011, 2012, 2013, and 2014. We picked 10 random papers from each conference year out of the total number of papers at HPDC (22, 23, 20, 21), SC (74, 100, 91, 83), PPoPP (26, 26, 26, 28), covering between 10% and 50% of the population. We only considered peer-reviewed technical papers and did not include invited talks, panels, workshops, or tutorials in this population. From here on, we anonymize the conferences as ConfA, ConfB, ConfC and we omit references to specific papers because our main focus lies on improving the discipline in our field, not to finger-point at specific venues or previous work. Papers that did not report performance numbers of real-world experiments (e.g., simulations, floating point error analyses, or theoretical works) are treated as not applicable; we had a total of 25/120 (20%) not applicable papers.

The results of our study are shown in Table 1. The upper part of the table describes our analysis of each paper's documentation regarding its experimental design (setup). We differentiate three major classes for describing hardware: compute, memory, and network; three major classes for describing software: compiler, runtime, and storage; and three major classes for describing benchmarks: input and software, measurement setup, and source code. We analyzed each paper and report our subjective opinion if the data provided in each class would support an unquestionable interpretation (or even reproduction) of the results using the symbol ✓. If a paper was not applicable, we marked all results as (·). When we determined that the interpretability of an experiment is not depending on details of a specific class (e.g., a shared memory experiment does not need to describe the network), we mark the class also with ✓. The lower part of the table summarizes each paper's data analysis methods which we discuss in detail in Section 3.

In general, most papers report details about the hardware but fail to describe the software environment. We find that the average paper's results are hard to interpret and easy to question because measurement and reporting methods are often not well described. While the median scores of ConfA and ConfC seem to be improving over the years, there is no statistically significant evidence for this. Our general impression is that the state of the art is poor; hard to interpret experiments are a serious problem for the scientific advancement of a field.

We note that this is not limited to HPC, as many researchers identify similar shortcomings in other fields (cf. [20,32,36]). For example, at the leading languages and compilers conference PLDI'11, 39 out of 42 papers reporting execution times did not mention uncertainty [58]. Manolescu et. al [39] show that out of 78 accepted papers at a leading database conference SIGMOD'08, 54 participated in a repeatability experiment and only 44 could be (partially) repeated independently. Zannier, Melnik, and Maurer [64] conclude that the quantity of empirical software engineering research at the leading conference ICSE is increasing but that soundness of empirical evaluations can be improved. Other experimental fields outside of computer science are also affected, for example Altman [1] comments on the suboptimal state of the art in medicine.

If supercomputing benchmarking and performance analysis is to be taken seriously, the community needs to agree on a common set of standards for measuring, reporting, and interpreting performance results.

[1]The raw data can be found on the LibSciBench webpage.

2.1 Common Best/Worst Practices

During our analysis, we identified several frequent problems. This first discussion aims to establish a set of common sense rules that experimenters should consider to improve the interpretability of their results. We discuss experimental design and reporting in more detail in Section 4.

2.1.1 Use Speedup with Care

Speedup often leads to confusion because the base case is not always clear. In its most basic form, speedup is used to compare two systems, for example, if system A is s times as fast as system B, then the speedup $s = \frac{T_A}{T_B}$, where T_X is the respective execution time on system X. The relative gain of system A over system B can be reported as $\Delta = s - 1$; if, for example, system A is 20% faster than system B for $s = 1.2$.

In parallel computing, speedup is usually used to compare the gain of using more than one processor. Here, it is often unclear if the comparison is performed with respect to the best possible serial implementation or the potentially worse parallel implementation executed on a single processor. Stating the comparison target clearly is important because better serial algorithms that cannot be parallelized efficiently often exist.

Furthermore, speedup itself is a rather meaningless measure because it will typically be higher on slow processors and/or less optimized codes (e.g., [41]). Thus, while speedup can be used as a dimensionless metric for the scaling of a single algorithm on a single computer, it cannot be used to compare different algorithms or different computers. In order to provide insights into the results, the performance of the single-processor case must be stated in absolute terms. We generally suggest that speedup should only be used in exceptional cases as it can almost always be replaced by lower bounds on execution time (e.g., an "ideal scaling" line in a plot cf. Figure 7(b)). We found 39 papers reporting speedups and 15 (38%) of them did not include the absolute performance of the base case.

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.*

A simple generalization of this rule implies that one should never report ratios without absolute values.

2.1.2 Report Units Unambiguously

We observed a general sloppiness in reporting results, for example, the long-standing confusion if MFLOPs indicates a rate or a count still remains. We recommend following the suggestions made by the PARKBENCH committee [23, 46] and denote the number of floating point operations as flop (singular and plural), the floating point rate as flop/s, Bytes with B, and Bits with b. More confusion stems from the use of base-2 or base-10 number qualifiers. Here we suggest to either follow the IEC 60027-2 standard and denote binary qualifiers using the "i" prefixes such as MiB for Mebibytes or clarify the base. We found that the majority of the papers report confusing or incorrect metrics; we only consider two of the 95 papers fully unambiguous.

2.1.3 Do not Cherry-pick

It is important for interpretability that all results are presented, even the ones that do not show the desired effect. We briefly discuss two forms of subset-selection that should be avoided:

Use the whole node Scaling experiments should always utilize all available resources. For example, a multithreaded application should be run on all available cores even if it stops scaling.

Use the whole benchmark/application If an application optimization is the center of a study then one should consider the whole application runtime and not only kernels. Furthermore, if existing benchmark suites are used, all benchmarks should be run. If not, then a reason should be presented (e.g., a compiler transformation for C code cannot transform all NAS benchmarks).

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

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.

3. ANALYZING EXPERIMENTAL DATA

Many of the research publications analyzed in Section 2 are not reproducible and hardly interpretable. Even worse, without providing information about the variability in the measurements, reported results may be well in the statistical noise. We believe that sloppy data analysis and missing statistical background constitutes a big part of the problem. Thus, we now discuss various techniques to analyze measurement data such that sound conclusions can be based on them.

3.1 Summarizing Results

Performance data is often collected from multiple measurements with different configurations. Summarizing these results can be important because a single number, such as "our system runs twice as fast", is easiest to understand. Unfortunately, even leading researchers commonly make mistakes when summarizing data [2]. Thus, researchers need to be careful and precise about the techniques used to summarize results. We now provide an overview of different algebraic and statistical summary methods and how they should and should not be applied.

A major differentiating factor for summarizing is whether the results are deterministic or of a statistical nature. Deterministic measurements (e.g., flop count) can be summarized with simple algebraic methods while nondeterministic measurements (e.g., runtime) requires statistical techniques.

3.1.1 Algebraic Summaries of Deterministic Data

Algebraic summaries should only be used if there is no variance in the result measurements. Such deterministic measurements could be the number of instructions (e.g., flop or load/store) to compute a specific problem or, in certain (uncommon) scenarios, also execution times.

Summarizing costs Execution time is often the best metric for accessing computer performance because it has an undebatable meaning [53]. Other metrics that have a direct semantic, such as energy consumption, purchasing cost, or number of instructions, fall in the same category that we call *costs*. Costs typically have an atomic unit such as seconds, watts, dollars, or flop and their influence is linear. This enables statements such as "system A is twice as fast as system B". In the standard case *where all measurements are weighted equally use the arithmetic mean to summarize costs*: $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$.

Summarizing rates Cost metrics are often used to derive other measures such as flop/s, flop/watt, or flop/inst, to express cost rates. In this case, the arithmetic mean must not be used as it leads to the wrong conclusions [19, 38]. In general, *if the denominator has the primary semantic meaning, the harmonic mean provides correct results*: $\bar{x}^{(h)} = \frac{n}{\sum_{i=1}^{n} 1/x_i}$. If the absolute counts (e.g., flops and seconds) are available we recommend using the arithmetic mean for both before computing the rate.

Rule 3: *Use the arithmetic mean only for summarizing costs. Use the harmonic mean for summarizing rates.*

Summarizing ratios Both costs and rates have units. However, it is sometimes beneficial to argue in unit-less terms such as speedup ratios or percentage of system peak (cf. Section 5.1). In general, *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 [53].

The *geometric mean* $\bar{x}^{(g)} = \sqrt[n]{\Pi_{i=1}^{n} x_i}$ has been suggested for ratios. It is always lower than or equal to the arithmetic mean and always higher than or equal to the harmonic mean [22] and is thus often interpreted as the "more honest" mean. But in general, *the geometric mean has no simple interpretation and should thus be used with greatest care*. It can be interpreted as a log-normalized average as we describe in the next section. If in exceptional situations, for example, the cost measures are not available, normalized results have to be averaged, then they should be averaged using the geometric mean [19] because it provides better results but is strictly seen still incorrect [53].

In our review, 51 out of 95 applicable papers use summarizing to present results. Only four of these specify the exact averaging method. Numerous papers reported averages of either rates or ratios without specifying how they summarize, leaving it up to the reader to guess. Only one paper correctly specifies the use of the harmonic mean. Two papers report that they use the geometric mean, both without a good reason.

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.*

HPL example Let us assume we ran an HPL benchmark that needs 100 Gflop three times and measured the execution times $(10, 100, 40)$ seconds. The arithmetic mean of the execution times is 50s, indicating an average floating point rate of 2 Gflop/s, which would be the observed rate for a consecutive run of all three benchmarks. The arithmetic mean of the three rates would be 4.5 Gflop/s, which would not be a good average measure as it focuses on an individual run. The harmonic mean of the rates returns the correct 2 Gflop/s. Assuming the peak floating point rate is 10 Gflop/s, the three experiments would run at relative rates of $(1, 0.1, 0.25)$ and the geometric mean would be 0.29, indicating an (incorrect) efficiency of 2.9 Gflop/s. Note that we present this example purely for illustrative purposes, such nondeterministic measurements would need to be analyzed statistically as we explain in the next section.

3.1.2 Statistics of Normally Distributed Data

It has been recognized that performance varies significantly on today's supercomputers [63]. In fact, this problem is so severe that several large procurements specified upper bounds on performance variations as part of the vendor's deliverables. In such nondeterministic systems, a single number often does not represent the actual performance well and interpretable reporting requires some measure of spread or stability of the results. The most general issue is to determine the statistical method to use. We differentiate between parametric and nonparametric techniques. If the distribution function is known, parametric techniques are most powerful, however, if the function is unknown, nonparametric techniques should be used. Some real-world measurements are normally distributed in which case analysis can be improved.

Restrictions Most of the statistics described in this section can only be used if measurements are independent samples of a normal distribution. Furthermore, these statistics assume that the arithmetic mean is the correct measure, thus, they cannot directly be applied to rates or ratios.

Standard deviation A simple measure of the spread of normally distributed samples x_i is the sample standard deviation $s = \sqrt{(\sum_{i=1}^{n}(x_i - \bar{x})^2)/(n-1)}$. It has the same unit as the (arithmetic) mean[2] and can be directly compared. It can be computed incrementally (online) through the sample variance $s_n^2 = (n-2)/(n-1)s_{n-1}^2 + (x_n - \bar{x}_{n-1})/n$ and the required sample mean $\bar{x}_n$ can be computed online using $\bar{x}_n = \bar{x}_{n-1} + (x_n - \bar{x}_{n-1})/n$. We note that these schemes can be numerically unstable and more complex stable schemes may need to be employed for large numbers of samples.

Coefficient of Variation The coefficient of variation is a related dimensionless metric that represents the stability of a set of normally distributed measurement results: $\text{CoV} = s/\bar{x}$. It has been demonstrated as a good measure for the performance consistency of a system over longer periods of time [34, 52].

Confidence intervals of the mean The arithmetic mean can be a good summary if the result is used to predict the sum of a set of measurements (e.g., the execution time of a number of loop iterations). However, $\bar{x}$ is computed from a sample of the data and may not represent the true mean well. Confidence Intervals (CIs) are a tool to provide a range of values that include the true mean with a given probability depending on the estimation procedure. To determine a CI, we select a confidence value $1-\alpha$ (typically 0.99, 0.95, or 0.9) and find two probabilistic bounds, $b_1 < \bar{x} < b_2$ around $\bar{x}$ such that $\Pr[b_1 \leq \mu \leq b_2] = 1-\alpha$. Here, μ is the true mean.

We assume that the exact standard deviation is unknown and we only have the mean and deviation of the samples. Thus, our calculation bases on Student's t distribution with $n-1$ degrees of freedom, returning the CI: $[\bar{x} - t(n-1, \alpha/2)s/\sqrt{n}, \bar{x} + t(n-1, \alpha/2)s/\sqrt{n}]$ where $t(n-1, p)$ can be obtained from a table and converges towards the standard normal distribution for large n.

CIs are often misunderstood; they do neither provide a probability that any of the sample data or data of future experiments lies within the CI nor that the mean of future experiments lies within the interval. Instead, CIs provide a measure of reliability of the experiment. For example, a 95% CI is interpreted as a 95% probability that the observed CI contains the true mean. *The CI represents a frequentist view with a fixed true mean and variable (random) bounds* while the so called 95% *credible interval* provides a Bayesian view with fixed bounds and a variable mean which estimates that the mean is with a 95% chance in the interval. We do not discuss credible intervals here because they require assumptions on the prior distribution of the mean. Most of the papers in our survey report nondeterministic data but only 15 mention some measure of variance (we also counted common statements like "the observed variance was low"). Only two out of 95 papers report confidence intervals around the mean.

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

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.*

Testing for Normality and Normalization.

Multiple tests can be used for checking if a dataset is normally distributed. Razali and Wah [48] showed empirically that the Shapiro-Wilk test [51] is most powerful, yet, it may be misleading for large sample sizes. We thus suggest to check the test result with a Q-Q plot or an analysis specific to the used statistics.

Log-normalization Many nondeterministic measurements that are always positive are skewed to the right and have a long tail following a so called log-normal distribution. Such measurements can

[2] if not otherwise stated, mean refers to arithmetic mean

be normalized by transforming each observation logarithmically. Such distributions are typically summarized with the log-average, which is identical to the geometric mean: $\bar{x}^{(g)} = \exp\left[\frac{1}{n}\sum_{i=1}^{n} \ln x_i\right]$.

Normalization If the data is not normally distributed, it can be normalized by averaging intervals of length k until the result is normal (following the Central Limit Theorem (CLT)). However, this technique loses precision, as one cannot make any statements about the individual measurements, and it is not guaranteed that any realistic k will suffice for normalization. Normalization is demonstrated in Figure 2: The top-left part shows the density function

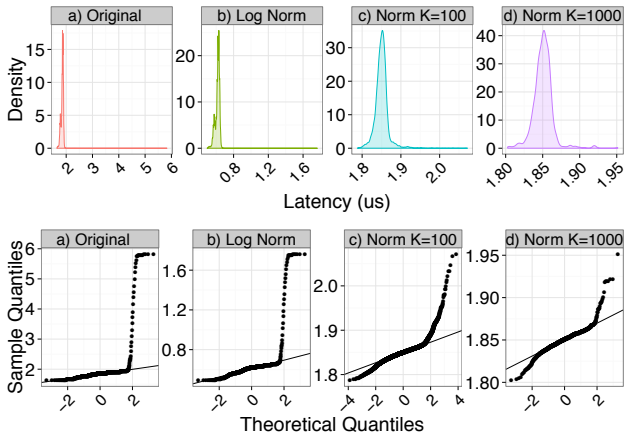


Figure 2: Normalization of 1M ping-pong samples on Piz Dora.

of a ping-pong latency benchmark with 64B on Piz Dora (cf. Section 4.1.2). Below this, we show a Q-Q plot which can be used to visually inspect the data for normality. It shows the relation between the quantiles of a standard normal distribution and the observed distribution; if the two should agree after linearly transforming the values, the points should lie on a straight line. Thus, the closer the plotted quantile relation is to a straight line, the more normal the distribution is. The remaining three plots in the top part of the figure show various normalization strategies with the corresponding Q-Q plots in the lower part.

Large number of samples and the mean The mean is a summed metric and thus, its distribution asymptotically converges towards a normal distribution (CLT). Thus, for large enough sample sizes, one can assume normality for most mean-based metrics. However, "large enough" depends on the details of the distribution. Our experiments (Figure 2) and other authors [12] show that the 30-40 samples as indicated in some textbooks [38] are not sufficient. We recommend attempting to normalize the samples until a test of the resulting distribution indicates normality or use the nonparametric techniques described in the next section.

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

Two papers in our survey use confidence intervals around the mean but authors do not seem to test the data for normality. The top of Figure 3 demonstrates how assuming normality can lead to wrong conclusions: the CI around the mean is tiny while the mean does not represent the distribution well.

3.1.3 Statistics of Non-normally Distributed Data

Normal distributions are often observed in natural and social sciences where independent sources of errors can add to and remove from the true quantity. However, *normal distributions are only rarely observed when measuring computer performance*, where most system effects lead to increased execution times. Sources of error are scheduling, congestion, cache misses etc., typically leading to multi-modal distributions that are heavily skewed to the right.

Restrictions Nonparametric metrics, such as the median or other percentiles, do not assume a specific distribution and are most robust. However, these measures also require independent and identically distributed (iid) measurements.

Median and quartiles The median of n measurements is the measurement at position (or *rank*) $n/2$ after sorting all data. The median is also called the 50th percentile denoted as $x_{(50\%)}$. The lower and upper quartiles, or equivalently the 25th and 75th percentile, are at rank $n/4$ and $3n/4$. Rank measures are more robust with regards to outliers but do not consider (include) all measured values and are harder to interpret mathematically. The quartiles provide information about the spread of the data and the skew. Different percentiles can be reported with an obvious interpretation. For example, saying that the 99th percentile of measured execution times is less than 3 seconds means that at least 99% of all measurement results took at most 3 seconds.

Confidence intervals of the median The width of any CI depends on the variability and the number of measurements. In contrast to CIs around the mean that assume a normal distribution, one cannot calculate standard errors. However, nonparametric CIs can be computed and have a similar interpretation. These CIs may be asymmetric if the modeled distribution is asymmetric.

Le Boudec [9] shows that the $1-\alpha$ CI ranges from the measurement at rank $\left\lfloor \frac{n-z(\alpha/2)\sqrt{n}}{2} \right\rfloor$ to rank $\left\lceil 1 + \frac{n+z(\alpha/2)\sqrt{n}}{2} \right\rceil$ where $z(p)$ represents the normal distribution that can be found in tables (e.g., [9, App. A]). For example, for a 95% CI, $z(0.025) = 1.96$. We note that one cannot compute exact CIs because it only considers measured values as ranks and the resulting bounds can be slightly wider than necessary (in practice, this effect is usually negligible). Confidence intervals for other percentiles can be computed similarly [9].

On Removing Outliers.

Outliers can impact calculations of statistical measures such as mean or standard deviation. There is no rigorous mathematical definition which values should be seen as an outlier and such a definition would always depend on the context. We suggest to avoid removal of outliers and instead use robust measures such as percentiles. However, if outliers must be removed (e.g., if the mean is the required measure, cf. Section 3.1.2) then we recommend using Tukey's method which classifies values that are outside the interval $[x_{(25\%)} - 1.5(x_{(75\%)} - x_{(25\%)}), x_{(75\%)} + 1.5(x_{(75\%)} - x_{(25\%)})]$. We remark that one can increase Tukey's constant 1.5 in order to be more conservative. In any case, one should report the number of removed outliers for each experiment.

3.2 Comparing Statistical Data

We know how to derive estimates for the mean and median and CIs for the reliability of our experiments for each. However, due to the statistical nature of the measurements, special care needs to be taken when comparing values because differences in median or mean may not be *statistically significant*.

The simplest method is to compare confidence intervals. If $1-\alpha$ confidence intervals do not overlap, then one can be $1-\alpha$ confident that there is a statistically significant difference. The converse is not true, i.e., overlapping confidence intervals may still allow the difference to be statistically significant.

3.2.1 Comparing the Mean

To compare two means with one varying variable, one could use the t-test to compare two experiments or the one-factor analysis of

variance (ANOVA) which generalizes the t-test to k experiments. Both *require iid data from normal distributions with similar standard deviations*. ANOVA can also be used to compare multiple factors but this is rare in performance measurements: typically, one compares the effect of an optimization or system on various applications. The default null hypothesis assumes equality of all means and must be rejected to show statistical significance.

ANOVA considers k groups of n measurements each and computes the F test statistic as $F = egv/igv$ where $egv = \sum_{i=1}^{n} n(\bar{x}_i - \bar{x})^2/(k-1)$ represents the inter-group variability and $igv = \sum_{i=1}^{k}\sum_{j=1}^{n}(x_{ij} - \bar{x}_i)^2/(nk-k)$ represents the intra-group variability. The value $\bar{x}$ represents the overall sample mean of the measurements and $\bar{x}_i$ the sample mean of the ith group. The computed F ratio needs to exceed $F_{crit}(k-1, nk-k, \alpha)$ to reject the null hypothesis. None of our investigated papers uses statistical arguments to compare results.

3.2.2 Comparing the Median

The nonparametric Kruskal-Wallis one-way ANOVA test [35] can be used to test if the medians of experiments following non-normal distributions with one varying factor differ significantly. The null hypothesis is that all medians are the same. If no two measurements have exactly the same value, we can compute $H = 12/(kn(kn+1))\sum_{i=1}^{k} n\bar{r}_i - 3(kn+1)$ with $\bar{r}_i = \sum_{i=0}^{n} r_{ij}/n$ and r_{ij} being the rank of observation j from group i among all observations. For large n, the significance can be assessed by comparing with a tabulated $\chi^2(k-1,\alpha)$ distribution. Kruskal and Wallis provide tables for $n < 5$ [35]. None of the 95 analyzed papers compared medians in a statistically sound way. Figure 3 shows two distributions with significantly different medians at a 95% confidence level even though many of the 1M measurements overlap.

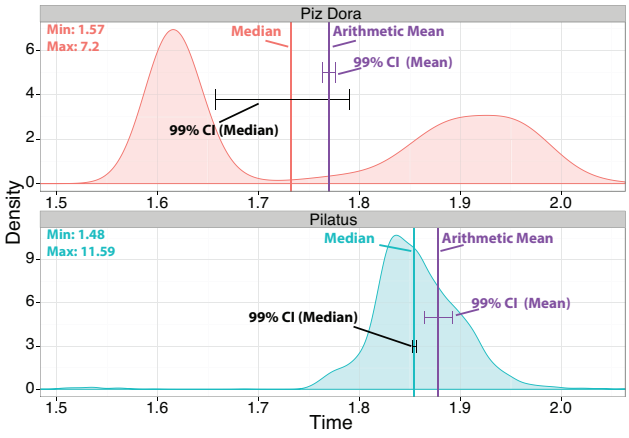


Figure 3: Significance of latency results on two systems.

Effect Size It has been shown that the tests described above can provide incorrect results for small effects [29,37,55]. Thus, we recommend using the effect size instead. The effect size expresses the differences between estimated means in two experiments relative to the standard deviation of the measurements: $\mathcal{E} = (\bar{X}_i - \bar{X}_j)/\sqrt{igv}$. We refer to Coe [13] for a more extensive treatment.

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

3.2.3 Quantile Regression

Quantile regression [33] is a powerful method for modeling the effect of varying a factor on arbitrary quantiles (which includes the median). It is a nonparametric measure that can be used to look at extrema. For example, for latency-critical applications, the 99th percentile is often more important than the mean. Oliveira et al. show that quantile regression can lead to deep insights into measurement data [16]. Quantile regression (QR) allows us to compare the effect across various ranks and is thus most useful if the effect appears at a certain percentile. Typically, quantile regression results are plotted with the quantiles on the x-axis, while the mean would present a single value with a confidence interval.

Quantile regression can be efficiently computed using linear programming [33], which is supported by many statistics tools. Figure 4 shows how the different quantiles of the latency of the Pilatus and Piz Dora systems. Piz Dora is used as the base for comparison and the "Intercept" shows latency as function of the percentiles (dots with 95% confidence level) and the mean with a 95% confidence interval (straight and dotted lines). The lower part shows a similar quantile plot with the difference to the Dora system. The difference of the means is $0.108\mu s$. The interesting observation in the QR plot is that low percentiles are significantly slower on Piz Dora than on Pilatus while high percentiles are faster. So for bad-case latency-critical applications Pilatus would win even though median and mean indicate the opposite (statistically significant).

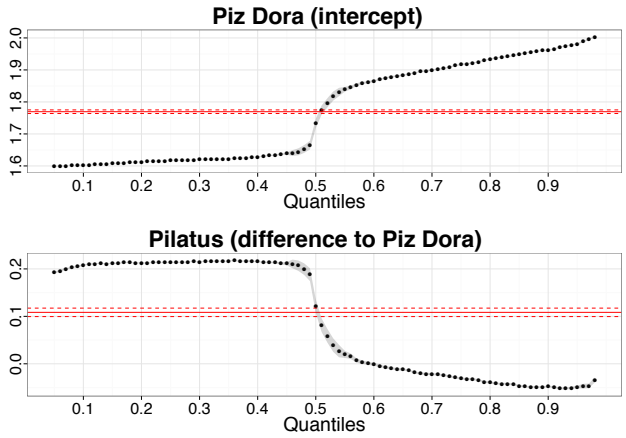


Figure 4: Quantile regression comparison of the latencies comparing Pilatus (base case or intercept) with Piz Dora.

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.*

4. EXPERIMENTAL DESIGN

Good experimental design is imperative for empirical sciences. Thus, several books describe general techniques to design reproducible (and thus interpretable) experiments. We recommend *factorial design* to compare the influence of multiple factors, each at various different levels, on the measured performance. This allows experimenters to study the effect of each factor as well as interactions between factors. We refer to standard textbooks [6,10,42] for a description of such basic experimental design techniques.

Here, we describe specifics of benchmarking on parallel systems that are not covered in the general techniques. Specifically, we discuss how to ensure interpretability, measure and collect data, controlling the experimental setup, and rules for data analysis. We focus on measuring *intervals* (e.g., time, energy, cost, or various counts) of applications or microbenchmarks. Other techniques such as tracing or profiling are outside the scope of this paper.

4.1 Designing Interpretable Measurements

The major goal of experimental design for scientific work is to ensure reproducibility or at least interpretability. This requires that

all significant setup parameters are carefully selected and clearly documented. The setup determines the elements that are varying experimental factors and the fixed environment. The evaluate collaboratory [7] collects common patterns of non-reproducibility in computer science research. We believe that it is impossible to design a complete set of rules because these will always be limited to the particular research context; here, we discuss common issues specific to parallel computing.

4.1.1 Measurement Environment

It is often desired to set up a fixed environment with as little variation as possible in order to focus on a specific factor. For example, when conducting scalability experiments, one would like to only vary the number of CPUs while keeping other parameters constant. In general, the experimenter needs to consider parameters such as the allocation of nodes, process-to-node mappings, network or on-node interference, and other system effects that can possibly influence the outcome of the experiment. Fixing each of these parameters may or may not be possible, depending on the setup of the experimental system. If controlling a certain parameter is not possible then we suggest randomization following standard textbook procedures. For example, Hunold et al. [27] randomly change the execution order within a call to the job launcher. Such random experiments require to model the randomized parameter as a non-deterministic element.

4.1.2 Documenting the Setup

Reproducing experiments in HPC is notoriously hard because hardware is often specialized and impossible to access for the majority of scientists. Most fixed parameters, such as software, operating system versions, and compiler flags, that need to be reported are obvious. However, others, such as exact specifications of randomized inputs or even sharing complete random datasets may be forgotten easily. The common lack of system access makes it important that the hardware is described in a way that allows scientists to draw conclusions about the setting. For example, details of the network (topology, latency, and bandwidth), the connection bus of accelerators (e.g., PCIe), or the main memory bandwidth need to be specified. This enables simple but insightful back of the envelope comparisons even if the exact setting cannot be reproduced. Furthermore, batch system allocation policies (e.g., packed or scattered node layout) can play an important role for performance and need to be mentioned. These are just some examples and good reporting needs to be designed on a case-by-case basis.

We observed how several authors assumed that mentioning a well-known system such as NERSC's Hopper or ORNL's Titan is sufficient to describe the experimental setup. This is bad practice for several reasons: (1) regular software upgrades on these systems likely change performance observations, (2) it may be impossible for other scientists to determine the exact system parameters, and (3) implicit assumptions (e.g., that IBM Blue Gene systems are noise-free) are not always understood by all readers.

Warmup Some programs (especially communication systems) establish their working state on demand. Thus, to measure the expected time, the first measurement iteration should be excluded from the average computation. It will not affect the median or other ranks if enough measurements are taken to reach a tight CI.

Warm vs. cold cache It is important to consider the state of the system when the measurement is performed. One of the most critical states regarding performance is the cache. If small benchmarks are performed repeatedly, then their data may be in cache and thus accelerate computations. This may or may not be representative for the intended use of the code. Whaley and Castaldo [59] show the impact of cache on measurements of linear algebra codes and discuss how to flush caches.

Our experimental setup Each node of Piz Daint (Cray XC30) has an 8-core Intel Xeon E5-2670 CPU with 32 GiB DDR3-1600 RAM, an NVIDIA Tesla K20X with 6 GiB GDDR5 RAM, and uses Cray's Programming Environment version 5.1.29. Each node of Piz Dora (Cray XC40) has two 12-core Intel Xeon E5-2690 v3 CPUs with 64 GiB DDR4-1600 RAM and uses Cray's Programming Environment version 5.2.40. Both systems are interconnected by Cray's Aries interconnect in a Dragonfly topology. Pilatus has two 8-core Intel Xeon 5-2670 CPUs with 64 GiB DDR3-1600 RAM per node, is connected with an InfiniBand FDR fat tree topology, and runs MVAPICH2 version 1.9. All ping-pong results use two processes on different compute nodes. For HPL we chose different allocations for each experiment; all other experiments were repeated in the same allocation. Allocated nodes were chosen by the batch system (slurm 14.03.7). The filesystem configuration does not influence our results. All codes are compiled with gcc version 4.8.2 using -O3.

4.2 How to Measure

After deciding on the factor(s) to investigate and fixing the environment parameters, researchers need to determine the levels (values) of each factor. For example, for a scalability study, they need to choose the numbers of processes to run the experiment with. This again depends on the desired use of the results. It is well known that several implementations perform better with 2^k, $k \in \mathbb{N}$ processes than with $2^k + 1$ processes, cf. Figure 5. It needs to be determined if the experiment should only show performance for powers-of-two process counts or the general case. This also applies to other special cases for arbitrary application inputs.

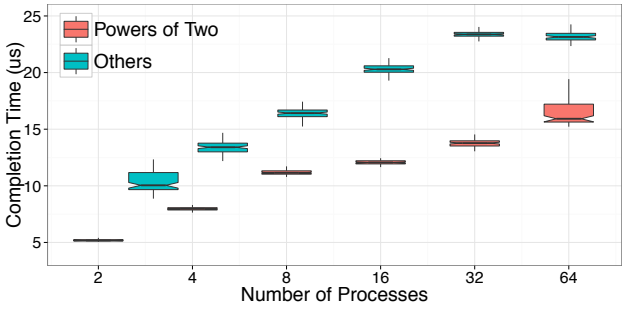


Figure 5: 1,000 MPI_Reduce runs for different process counts.

Weak and Strong Scaling Papers should always indicate if experiments are using strong scaling (constant problem size) or weak scaling (problem size grows with the number of processes). Furthermore, the function for weak scaling should be specified. Most commonly, the input size is scaled linearly with the number of processes. However, for non-work conserving algorithms, linear scaling can be misleading and more appropriate functions should be used. Also, when scaling multi-dimensional domains, papers need to document which dimensions are scaled because, depending on the domain decomposition, this could cause significant performance differences (e.g., if not all dimensions are distributed).

Adaptive Level Refinement Our example demonstrates that the levels of each factor must be selected carefully. With certain assumptions on the parameters, one could use adaptive refinement to measure levels where the uncertainty is highest, similar to active learning [50]. SKaMPI [49] uses this approach assuming parameters are linear.

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.*

This rule is more generic as the experimental design and reporting needs to be tailored to the specific experiment. Ideally, researchers release the source code used for the experiment or at least the input data and their generators.

4.2.1 Measuring Time in Parallel Systems

After factors and levels have been decided, one needs to choose a measurement mechanism. We only describe time and assume that other mechanisms (e.g., energy) require similar considerations. We distinguish between two types of measurements: (1) *full benchmarks* which are executed multiple times in a controlled environment and (2) *kernel benchmarks* where parts of applications are measured during execution. Full benchmarks (e.g., microbenchmarks) can be started synchronously while kernel benchmarks are executed by the application control flow. Whaley and Castaldo [59] discuss common issues when timing serial systems; we extend the discussion to parallel systems.

Measurement perturbances Measuring run times induces overheads for reading the timer, and so researchers need to ensure that the timer overhead is only a small fraction (we suggest <5%) of the measurement interval. Furthermore, researchers need to ensure that the timer's precision is sufficient to measure the interval (we suggest 10x higher). Microbenchmarks can be fully controlled and multiple iterations can be measured in a single interval, thus, timer overhead and precision is less important. For kernel benchmarks, timer precision and overhead limit the smallest measurable interval. It is important to consider the impact of system noise in the experimental design where small perturbations in one process can propagate to other processes.

Measuring multiple events Microbenchmarks can simply be adapted to measure multiple events if the timer resolution or overhead are not sufficient. This means to measure time for k executions and compute the sample mean $\bar{x}^k = T/k$ and repeat this experiment n times and compute statistics over n samples. However, this loses resolution in the analysis: one can no longer compute the confidence interval for a single event. Furthermore, rank measures such as percentiles and median can only be computed for blocks of k measurements. Thus, we recommend measuring single events to allow the computation of confidence intervals and exact ranks.

Parallel time 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 [25].

Many evaluations use an (MPI or OpenMP) barrier to synchronize processes for time measurement. This may be unreliable because neither MPI nor OpenMP provides timing guarantees for their barrier calls. While a barrier commonly synchronizes processes enough, we recommend checking the implementation. For accurate time synchronization, we propose to use the simple delay window scheme [25,62]. In this scheme, a master synchronizes the clocks of all processes and broadcasts a common start time for the operation. The start time is sufficiently far in the future that the broadcast will arrive before the time itself. Each process then waits until this time and the operation starts synchronously.

Summarize times across processes After measuring n events on P processes the question arises how to summarize the nP values. This depends on the purpose of the summary; sometimes a simple minimum or maximum across the processes seems sufficient. We recommend avoiding such non-robust measures. Instead, we recommend performing an ANOVA test to determine if the timings of different processes are significantly different. If the test indicates no significant difference, then all values can be considered from the same population. Otherwise, more detailed investigations may be necessary in order to choose a suitable summary. Common summaries across processes are maximum or median. Figure 6 shows the measured timings of 1,000 reductions with 64 processes on the Piz Daint system with a significant difference for some processes.

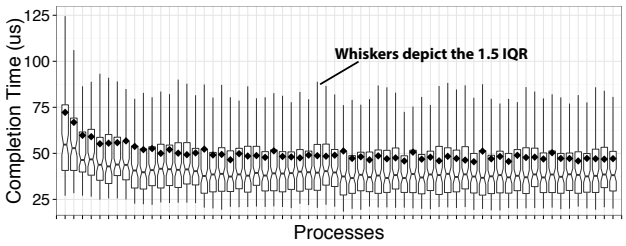


Figure 6: Variation across 64 processes in MPI_Reduce.

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

For example, we plot the maximum across processes for the reduction time in Figure 5 to assess worst-case performance.

4.2.2 Number of Measurements

Measurements on supercomputers can be expensive, especially if they require the full machine. Thus, careful planning of the number of needed measurements is needed. At the same time, one needs to guarantee enough measurements in order to make statistically sound conclusions. We have seen that the CI is a good measure for the confidence in the results. We now show how the CI can be used to compute the number of measurements needed to achieve a specified error certainty in the result. The error certainty is expressed by two values: (1) the confidence level $1-\alpha$ which we know from CIs, and (2) the allowed error level $1-e$ relative to the mean or median.

Normally distributed data For a normal distribution, one can compute the number of required measurements based on previous measurements. The acceptable error $1-e$ (e.g., 10%) defines the confidence interval $[\bar{x} - e\bar{x}, \bar{x} + e\bar{x}]$. A simple manipulation of the CI equations (cf. Section 3.1.2) results in $n = \left(\frac{s \cdot t(n-1,\alpha/2)}{e\bar{x}}\right)^2$.

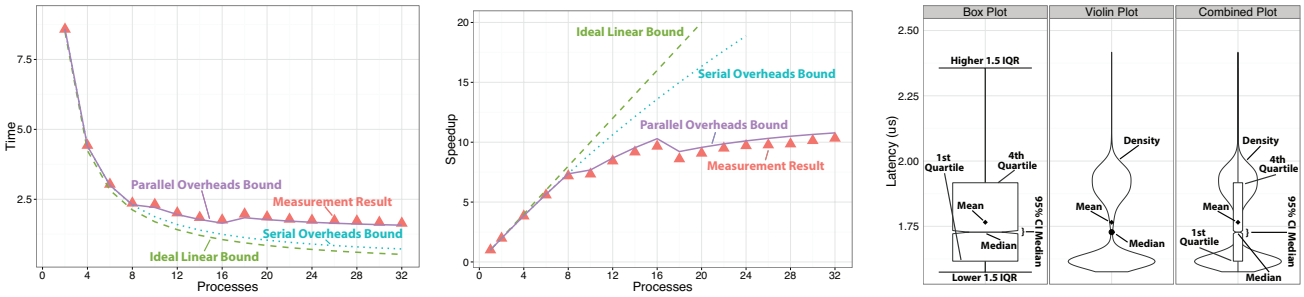
Non-normally distributed data If the distribution is unknown, one cannot easily compute the required number of measurements analytically. However, it is possible to check if a given set of measurements satisfies the required accuracy. Thus, we recommend recomputing the $1-\alpha$ CI (cf. Section 3.1.3) after each $n_i = i \cdot k$, $i \in \mathbb{N}$ measurements and stop the measurement once the required interval is reached. We recommend choosing k based on the cost of the experiment, e.g., $k = 1$ for expensive runs. Furthermore. we note that $n > 5$ measurements are needed to assess confidence intervals nonparametrically.

5. REPORTING RESULTS

Correctly communicating experimental results is at least as important as rigorous measurements. The communication should facilitate two purposes: (1) provide insight and intuition of the behavior of the system (interpretability) and (2) enable reproduction by the community.

5.1 Simple Bounds Modeling

Ullmann [57] argues that experiments alone have limited value as a validation for research because experiments are limited to specific and often small sets of variants and can thus not represent the



(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.

general case. Conversely, analytic performance models are most general but maybe inaccurate because they have to ignore many details of the system. We suggest to combine simple analytic or semi-analytic modeling with rigorous measurements in order to put the results into perspective [24]. For example, a simple but powerful form of modeling is to consider the achieved performance relative to an upper bound. Historically, this is done relative to the achievable peak floating point performance [23]. Recently, floating point performance has become less important and other system parameters, such as memory or network bandwidth limit performance.

In general, we can model any computer system's capabilities as a k-dimensional space where each dimensions represents one particular functionality or *feature* of the machine. We denote a specific machine model by $\Gamma = (p_1, p_2, \ldots, p_k)$. Features are typically expressed as rates and each p_i represents the maximum achievable performance of that feature on the machine. We assume that achieved performance of each feature can be measured, for example, an application's floating point rate or memory bandwidth. Thus, an application measurement can be represented as $\tau = (r_1, r_2, \ldots, r_k)$ and $r_i \leq p_i$. An application's performance can now be expressed in terms of the peak rates as a dimensionless metric $\mathcal{P} = (r_1/p_1, r_2/p_2, \ldots, r_k/p_k)$.

Such a normalized view has several advantages, for example, it is easy to determine which (set of) feature(s) is a likely bottleneck. The application's requirements vector can be used to determine the *balancedness* of a machine for a specific program or a set of programs. It can even be employed to proof optimality of a given implementation if (1) r_j/p_j is close to one *and* (2) one can show that the application cannot be solved with less operations of the jth feature. It is advisable to limit the set of features to the most significant ones, typical features are memory bandwidth and floating point rates as popularized by the roofline model ($k = 2$) [60]. Sometimes, analytical upper bounds for Γ are far from reality (the vendor-specified numbers are only guarantees to not be exceeded). In these cases, one can parametrize the p_i using carefully crafted and statistically sound microbenchmarks.

For parallel codes, one could model the scalability as a feature, however, this requires special care. We distinguish three cases of bounds models with growing complexity:

Ideal linear speedup The simplest upper performance bound is that p processes cannot speed up calculations more than p times (ideal scaling). Super-linear scaling which has been observed in practice is an indication of suboptimal resource use for small p.

Serial overheads (Amdahl's law) If the fraction of parallelized code b is known, then one can show a tighter upper bound based on Amdahl's law. Here, the speedup is limited to $(b + (1 - b)/p)^{-1}$.

Parallel overheads Some parallel operations cause overheads that grow with the number of processes, for example, a reduction operation cannot execute faster than $\Omega(\log p)$. If the parallelization overheads are known, then one can specify a tighter upper bound which can be combined with Amdahl's law. Figure 7 shows scaling results from calculating digits of Pi on Piz Daint. The code is fully parallel until the execution of a single reduction; the base case takes 20ms of which 0.2ms is caused by a serial initialization (b=0.01). The three lines show the bounds for ideal speedup, serial overheads, and parallel overheads. The parallel overheads assume the following (empirical) piecewise model for the final reduction: $f(p \leq 8) = 10ns$, $f(8 < p \leq 16) = 0.1ms \cdot \log_2(p)$, $f(p > 16) = 0.17ms \cdot \log_2(p)$ (the three pieces can be explained by Piz Daint's architecture). The parallel overhead bounds model explains nearly all the scaling observed and provides highest insight.

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

5.2 Graphing Results

"Use a picture. It's worth a thousand words." (1911). Indeed, there are many guidelines and textbooks for graphing information exist [18, 56]. Here, we focus on graphing techniques that we find useful to communicate measurement results in parallel computing. Producing interpretable and informative graphics is an art that needs to be adapted to each specific case. For example, the choice between line-plots, histograms, boxplots, and violin plots depends on the number of levels and factors to show. Here, we show some general guidelines for advanced plotting techniques.

Box plots An example box plot is shown in the left side of Figure 7(c). Box plots [40] offer a rich set statistical information for arbitrary distributions: the box indicates the 25% (lower border) and the 75% (upper border) percentile. The middle bar denotes the median (50% percentile). The optional whiskers can plot different metrics such as min and max observations, various percentiles (90%, 99%), or the lowest observation in the 1.5 inter-quartile-range (IQR) (cf. outliers). Thus, the semantics of the whiskers must be specified. Furthermore, notched boxplots indicate a range of statistical significance, similar to a confidence interval around the median (typically 95%). Thus, non-overlapping notches indicate significant differences.

Histograms and violin plots Histograms show the complete distribution of data. Similarly, violin plots, as shown in the middle of Figure 7(c), depict the density distribution for all observations. They also typically show the median as well as the quartiles, similar to the boxplot. Violin plots contain thus more information than box plots but require more horizontal space to enable interpretation. Box plots and violin plots can also be combined in a single plot as shown in the right part of Figure 7(c).

Plotting summary statistics Summary statistics such as mean

and median can be plotted in different ways, for example, geometric and arithmetic mean can be added to box and violin plots as shown in Figure 7(c). Typically means are plotted as points or bar charts. Points should only be connected if they indicate a trend and values between two points are expected to follow the line (e.g., scaling with process counts). Otherwise, bar charts may be more appropriate. We recommend reviewing Crowl [14] for a discussion of common mistakes when plotting summary statistics in parallel computing.

Plotting CIs Confidence intervals should be included in plots at each of the measurement points. In cases where the CI is extremely narrow and would only clutter the graphs, it should be omitted and reported in the text. Another possibility could be to plot different confidence intervals for different measurement and mark them in the legend (e.g., (*) for 90%, (**) for 95%, or (***) for 99%) as similar to p-values by Hunold et al. [27] (Figure 7).

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.*

This rule does not require to always plot confidence intervals or other notions of spread/error. If these are always below a bound and would clutter the graph, they can simply be stated in the describing text or figure caption.

6. LibSciBench

To facilitate the adoption of statistically sound measurements of parallel computations, we develop a C library called LibSciBench that automates many of the guidelines and processes described in this paper. A performance measurement library is more flexible than existing benchmark suites. It can be linked to any application to either measure execution times or it can be used as a building block for a new benchmark suite. The library seamlessly integrates with MPI and OpenMP applications and can easily be extended to new models of parallelism.

Timers LibSciBench offers high-resolution timers for many architectures (currently x86, x86-64, PowerPC, and Sparc). The library automatically reports the timer resolution and overhead on the target architecture. Furthermore, its integrated low-overhead data collection mechanism monitors the runtime overhead and provides warnings if it exceeds a certain level. LibSciBench has support for arbitrary PAPI counters.

Synchronization LibSciBench offers a window-based synchronization mechanism for OpenMP and MPI to synchronize processes based on real-time.

Data Analysis LibSciBench's low-overhead data collection mechanism produces datasets that can be read directly with established statistical tools such as GNU R. LibSciBench offers several R scripts to check for normality, compute CIs, perform ANOVA tests, and quantile regression. Furthermore, LibSciBench's R scripts support the generation of Q-Q plots, box plots, violin plots, and mixed plots.

7. RELATED WORK

Many efforts have focused on high-level issues such as a good selection of representative problems or programs for (parallel) computers [5, 46]; many of these are mentioned in our paper. Here, we provide a wider view. For example, Hockney [23] provides an overview of parallel computer benchmarking without considering detailed statistical techniques. Bailey provides common guidelines for reporting benchmarks in technical computing [4], following his humorous summary of the state of the art [3]. Several works (e.g., [21,27,49]) describe how to measure the performance of MPI functions accurately. Pakin defines various languages for performance testing and reproducibility [44, 45]. Other fields of experimental computer science struggle with similar issues and designed basic rules specific to their environment (e.g., [8, 32]). Stodden et al. [54] advocate a broad culture change to foster reproducibility in computational science. Hunold et al. show that the state of the art in the MPI community lacks [11]. They also discuss reproducibility of parallel computing research and suggest to adopt Drummond's "scientific replicability" [28]. We go quite a bit further and and simply advocate clear documentation to ensure interpretability. While many of these efforts have established valuable state of the practice, they did not come to a general consensus, and we so believe that it will be prohibitively hard to generate a complete checklist for experimental design in HPC.

We provide minimal guidelines for experimental design and focus on the more constrained topic of interpretable benchmarking where it seems easier to agree on a common set of techniques. More general guidelines (not specific to HPC) have been collected in the books of Lilja [38] and Jain [30]. However, some of their recommended data collection and reporting practices do not necessarily apply to HPC. More advanced statistical techniques such as bootstrap [15, 17] are beyond the scope of our work. A more general overview of experimental design and the scientific method is provided by Kirkup [31] or Wilson [61], respectively.

8. CONCLUSIONS

We present our first attempt to specify ground rules and guidelines for interpretable benchmarking of parallel systems. We intend to establish a minimal base for improving the quality in the field and hope to initiate a wider debate before the HPC community can adapt definitive guidelines. We do not suggest that our guidelines are complete, nor that they will solve all the problems associated with empirical research in parallel computing. However, this paper could serve as a reference for minimal quality in conducting and reporting experiments in parallel computing if all guidelines are followed carefully. For example, authors could ensure readers that they follow all rules and guidelines stated in this paper.

Exact reproduction of experiments on large parallel computers is close to impossible. Thus, we introduce the notion of interpretability which is a weaker form of reproducibility where a scientist can make his own conclusions or use the results in a similar setting. Ensuring interpretability requires to report as much information about nondeterministic measurements as needed for a statistically sound interpretation of the results. To facilitate the collection and reporting of interpretable data, we provide a portable and flexible measurement and data-analytics library that can be easily adapted to many experimental settings. Our library-based approach integrated with mature statistical analysis tools such as R saves time and improves the interpretability of benchmark results.

We feel strongly that, as a community, adopting these basic rules will both improve the quality of our research, and also the extent to which it can serve as a sound basis for future work.

Acknowledgments We thank Hans Rudolf Künsch, Martin Maechler, and Robert Gantner from the mathematics department at ETH Zurich for their invaluable feedback on the statistical parts of this paper. We thank David H. Bailey, William T. Kramer, Matthias Hauswirth, Timothy Roscoe, Gustavo Alonso, Georg Hager, Jesper Träff, and Sascha Hunold for feedback on early versions of this document. We thank Gilles Fourestier (CSCS) and Massimiliano Fatica (NVIDIA) for HPL setup and CSCS for providing compute resources. Finally, we thank the whole SPCL group, the SC15 anonymous reviewers, and everybody who contributed useful suggestions.

9. REFERENCES

[1] D. G. Altman. Statistical reviewing for medical journals. *Statistics in medicine*, 17(23):2661–2674, 1998.

[2] J. N. Amaral. How did this get published? Pitfalls in experimental evaluation of computing systems. *Languages, Compilers, Tools and Theory for Embedded Systems*, 2012.

[3] D. H. Bailey. Twelve ways to fool the masses when giving performance results on parallel computers. *Supercomputing Review*, pages 54–55, August 1991.

[4] D. H. Bailey. Misleading performance claims in parallel computations. In *Proceedings of the 46th Annual Design Automation Conference*, DAC '09, pages 528–533, New York, NY, USA, 2009. ACM.

[5] 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, et al. The NAS parallel benchmarks. *International Journal of High Performance Computing Applications*, 5(3):63–73, 1991.

[6] R. A. Bailey. *Design of Comparative Experiments*. Cambridge University Press, 2008.

[7] S. M. Blackburn et al. Can you trust your experimental results? Technical report, Evaluate Collaboratory, TR #1, February 2012.

[8] S. M. Blackburn, K. S. McKinley, R. Garner, C. Hoffmann, A. M. Khan, R. Bentzur, A. Diwan, D. Feinberg, D. Frampton, S. Z. Guyer, M. Hirzel, A. Hosking, M. Jump, H. Lee, J. E. B. Moss, A. Phansalkar, D. Stefanovik, T. VanDrunen, D. von Dincklage, and B. Wiedermann. Wake up and smell the coffee: Evaluation methodology for the 21st century. *Commun. ACM*, 51(8):83–89, Aug. 2008.

[9] J.-Y. L. Boudec. *Performance Evaluation of Computer and Communication Systems*. EPFL Press, 2011.

[10] G. E. Box, W. G. Hunter, and J. S. Hunter. *Statistics for experimenters: Design, Innovation, and Discovery, 2nd Edition*. Wiley-Interscience, 2005.

[11] A. Carpen-Amarie, A. Rougier, and F. Lübbe. Stepping stones to reproducible research: A study of current practices in parallel computing. In *Euro-Par 2014: Parallel Processing Workshops*, volume 8805 of *Lecture Notes in Computer Science*, pages 499–510. Springer International Publishing, 2014.

[12] T. Chen, Y. Chen, Q. Guo, O. Temam, Y. Wu, and W. Hu. Statistical performance comparisons of computers. In *Proceedings of the 2012 IEEE 18th International Symposium on High-Performance Computer Architecture*, HPCA '12, pages 1–12, Washington, DC, USA, 2012. IEEE Computer Society.

[13] R. Coe. It's the effect size, stupid: What effect size is and why it is important. 2002.

[14] L. Crowl. How to measure, present, and compare parallel performance. *Parallel Distributed Technology: Systems Applications, IEEE*, 2(1):9–25, Spring 1994.

[15] A. C. Davison and D. V. Hinkley. *Bootstrap Methods and their Application*. Cambridge University Press, October 1997.

[16] A. B. de Oliveira, S. Fischmeister, A. Diwan, M. Hauswirth, and P. F. Sweeney. Why you should care about quantile regression. *SIGARCH Comput. Archit. News*, 41(1):207–218, Mar. 2013.

[17] B. Efron and R. J. Tibshirani. *An Introduction to the Bootstrap*. Chapman & Hall, New York, 1993.

[18] S. Few. *Show Me the Numbers: Designing Tables and Graphs to Enlighten*. Analytics Press, June 2012.

[19] P. J. Fleming and J. J. Wallace. How not to lie with statistics: The correct way to summarize benchmark results. *Commun. ACM*, 29(3):218–221, Mar. 1986.

[20] A. Georges, D. Buytaert, and L. Eeckhout. Statistically rigorous java performance evaluation. *SIGPLAN Not.*, 42(10):57–76, Oct. 2007.

[21] W. Gropp and E. L. Lusk. Reproducible measurements of MPI performance characteristics. In *Proceedings of the 6th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 11–18, London, UK, UK, 1999. Springer-Verlag.

[22] P. W. Gwanyama. The HM-GM-AM-QM inequalities. *The College Mathematics Journal*, 35(1):47–50, January 2004.

[23] R. W. Hockney. *The Science of Computer Benchmarking*. Software, environments, tools. Society for Industrial and Applied Mathematics, 1996.

[24] T. Hoefler, W. Gropp, M. Snir, and W. Kramer. Performance Modeling for Systematic Performance Tuning. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC'11), SotP Session*, Nov. 2011.

[25] T. Hoefler, T. Schneider, and A. Lumsdaine. Accurately measuring collective operations at massive scale. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1–8, April 2008.

[26] T. Hoefler, T. Schneider, and A. Lumsdaine. Characterizing the influence of system noise on large-scale applications by simulation. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, pages 1–11, Washington, DC, USA, 2010. IEEE Computer Society.

[27] S. Hunold, A. Carpen-Amarie, and J. L. Träff. Reproducible MPI micro-benchmarking isn't as easy as you think. In *Proceedings of the 21st European MPI Users' Group Meeting*, EuroMPI/ASIA '14, pages 69:69–69:76, New York, NY, USA, 2014. ACM.

[28] S. Hunold and J. L. Träff. On the state and importance of reproducible experimental research in parallel computing. *CoRR*, abs/1308.3648, 2013.

[29] J. P. Ioannidis. Why most published research findings are false. *Chance*, 18(4):40–47, 2005.

[30] R. Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Wiley, 1991.

[31] L. Kirkup. *Experimental Methods: An Introduction to the Analysis and Presentation of Data*. John Wiley & Sons, February 1995.

[32] B. Kitchenham, S. Pfleeger, L. Pickard, P. Jones, D. Hoaglin, K. El Emam, and J. Rosenberg. Preliminary guidelines for empirical research in software engineering. *Software Engineering, IEEE Transactions on*, 28(8):721–734, Aug 2002.

[33] R. Koenker. *Quantile regression*. Cambridge University Press, May 2005.

[34] W. Kramer and C. Ryan. *Performance Variability of Highly Parallel Architectures*, volume 2659 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2003.

[35] W. H. Kruskal and W. A. Wallis. Use of ranks in one-criterion variance analysis. *Journal of the American*

statistical Association, 47(260):583–621, 1952.

[36] S. Kurkowski, T. Camp, and M. Colagrosso. Manet simulation studies: The incredibles. *SIGMOBILE Mob. Comput. Commun. Rev.*, 9(4):50–61, Oct. 2005.

[37] J. T. Leek and R. D. Peng. Statistics: P values are just the tip of the iceberg. *Nature*, 520(7549), Apr. 2015.

[38] D. J. Lilja. *Measuring computer performance: a practitioner's guide*. Cambridge University Press, 2000.

[39] I. Manolescu, L. Afanasiev, A. Arion, J. Dittrich, S. Manegold, N. Polyzotis, K. Schnaitter, P. Senellart, S. Zoupanos, and D. Shasha. The repeatability experiment of SIGMOD 2008. *ACM SIGMOD Record*, 37(1):39–45, 2008.

[40] R. McGill, J. W. Tukey, and W. A. Larsen. Variations of Box Plots. *The American Statistician*, 32(1):12–16, 1978.

[41] F. McSherry, M. Isard, and D. G. Murray. Scalability! but at what cost? In *15th Workshop on Hot Topics in Operating Systems (HotOS XV)*, Kartause Ittingen, Switzerland, May 2015. USENIX Association.

[42] D. C. Montgomery. *Design and Analysis of Experiments*. Wiley, 2012.

[43] T. Mytkowicz, A. Diwan, M. Hauswirth, and P. F. Sweeney. Producing wrong data without doing anything obviously wrong! *SIGPLAN Not.*, 44(3):265–276, Mar. 2009.

[44] S. Pakin. Conceptual: a network correctness and performance testing language. In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, pages 79–, April 2004.

[45] S. Pakin. The design and implementation of a domain-specific language for network performance testing. *IEEE Trans. Parallel Distrib. Syst.*, 18(10):1436–1449, Oct. 2007.

[46] PARKBENCH Committee/Assembled by R. Hockney (Chairman) and M. Berry (Secretary). PARKBENCH report: Public international benchmarks for parallel computers. 3(2):101–146, Summer 1994.

[47] F. Petrini, D. J. Kerbyson, and S. Pakin. The case of the missing supercomputer performance: Achieving optimal performance on the 8,192 processors of ASCI Q. In *Proceedings of the 2003 ACM/IEEE Conference on Supercomputing*, SC '03, pages 55–, New York, NY, USA, 2003. ACM.

[48] N. M. Razali and Y. B. Wah. Power comparisons of Shapiro-Wilk, Kolmogorov-Smirnov, Lilliefors and Anderson-Darling tests. *Journal of Statistical Modeling and Analytics 2*, (1):21–33, 2011.

[49] R. Reussner, P. Sanders, L. Prechelt, and M. Müller. Skampi: A detailed, accurate MPI benchmark. In *Proceedings of the 5th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 52–59, London, UK, UK, 1998. Springer-Verlag.

[50] B. Settles. *Active Learning*. Morgan & Claypool Publishers, July 2012.

[51] S. S. Shapiro and M. B. Wilk. An analysis of variance test for normality (complete samples). *Biometrika*, pages 591–611, 1965.

[52] D. Skinner and W. Kramer. Understanding the causes of performance variability in hpc workloads. In *Workload Characterization Symposium, 2005. Proceedings of the IEEE International*, pages 137–149, Oct 2005.

[53] J. E. Smith. Characterizing computer performance with a single number. *Commun. ACM*, 31(10):1202–1206, Oct. 1988.

[54] 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. Technical report, ICERM report, February 2013.

[55] D. Trafimow and M. Marks. Editorial. *Basic and Applied Social Psychology*, 37(1):1–2, 2015.

[56] E. Tufte. *The Visual Display of Quantitative Information*. Graphics Pr; 2nd edition, January 2001.

[57] J. D. Ullman. Experiments as research validation - have we gone too far?, July 2013.

[58] J. Vitek and T. Kalibera. Repeatability, reproducibility, and rigor in systems research. In *Proceedings of the Ninth ACM International Conference on Embedded Software*, EMSOFT '11, pages 33–38, New York, NY, USA, 2011. ACM.

[59] R. C. Whaley and A. M. Castaldo. Achieving accurate and context-sensitive timing for code optimization. *Software: Practice and Experience*, 38(15):1621–1642, 2008.

[60] S. Williams, A. Waterman, and D. Patterson. Roofline: An insightful visual performance model for multicore architectures. *Commun. ACM*, 52(4):65–76, Apr. 2009.

[61] B. Wilson. *An Introduction to Scientific Research*. Dover Publications, January 1991.

[62] T. Worsch, R. Reussner, and W. Augustin. On benchmarking collective MPI operations. In *Proceedings of the 9th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 271–279, London, UK, UK, 2002. Springer-Verlag.

[63] N. Wright, S. Smallen, C. Olschanowsky, J. Hayes, and A. Snavely. Measuring and understanding variation in benchmark performance. In *DoD High Performance Computing Modernization Program Users Group Conference (HPCMP-UGC), 2009*, pages 438–443, June 2009.

[64] C. Zannier, G. Melnik, and F. Maurer. On the success of empirical studies in the international conference on software engineering. In *Proceedings of the 28th International Conference on Software Engineering*, ICSE '06, pages 341–350, New York, NY, USA, 2006. ACM.