Statistical Performance Comparisons of Computers

Tianshi Chen, Qi Guo, Olivier Temam, Yue Wu, Yungang Bao, Zhiwei Xu, *Senior Member, IEEE*, and Yunji Chen

Abstract—As a fundamental task in computer architecture research, performance comparison has been continuously hampered by the variability of computer performance. In traditional performance comparisons, the impact of performance variability is usually ignored (i.e., the means of performance observations are compared regardless of the variability), or in the few cases directly addressed with t-statistics without checking the number and normality of performance observations. In this paper, we formulate a performance comparison as a statistical task, and empirically illustrate why and how common practices can lead to incorrect comparisons. We propose a non-parametric hierarchical performance testing (HPT) framework for performance comparison, which is significantly more practical than standard t-statistics because it does not require to collect a large number of performance observations in order to achieve a normal distribution of sample mean. In particular, the proposed HPT can facilitate quantitative performance comparison, in which the performance speedup of one computer over another is statistically evaluated. Compared with the HPT, a common practice which uses geometric mean performance scores to estimate the performance speedup has errors of 8.0 to 56.3 percent on SPEC CPU2006 or SPEC MPI2007, which demonstrates the necessity of using appropriate statistical techniques. This HPT framework has been implemented as an open-source software, and integrated in the PARSEC 3.0 benchmark suite.

Index Terms—Performance comparison, performance distribution, t-statistics, hierarchical performance testing

1 Introduction

A fundamental practice for researchers, engineers and information services is to compare the performance of two architectures/computers using a set of benchmarks. As trivial as this task may seem, it is well known to be fraught with obstacles, especially selection of benchmarks [1], [2], [3] and non-deterministic performance [4]. In this paper, we focus on the issue of non-deterministic performance. The variability can have several origins, such as stochastic architecture+software behavior [5], [6], performance observation bias [7], or even applications themselves. Non-deterministic performance can be easily observed by repeated performance observations. For instance, the geometric mean (GM) performance speedups, over an initial baseline run, of 10 subsequent runs of SPLASH-2 on a commodity computer (Linux OS, 4-core 8-thread Intel i7 920 with 6 GB DDR2 RAM) are 0.94, 0.98, 1.03, 0.99, 1.02, 1.03, 0.99, 1.10, 0.98, 1.01.

From a statistical viewpoint, the non-deterministic performance of a computer is a random variable obeying a certain probability distribution called *performance distribution*. Accordingly, measuring the performance of a computer can be viewed as a process that *statistically samples* the performance distribution of the computer. We argue that *the performance comparison of two computers can be statistically formulated as the comparison between their performance distributions based on (statistical) performance sampling*. By analyzing and summarizing the collected performance samples, one may claim that "whether one computer outperforms another", but reliability of such a comparison result may severely be hampered by the non-deterministic performance of computers. Even when two computers/architectures have fairly different performance, such non-determinism can still make the comparison (e.g., estimating the performance speedup of one computer over another) confusing. Incorrect comparisons can, in turn, affect research or acquisition decisions, so the consequences are significant.

A scientific way of addressing the impact of non-deterministic performance is to compute the confidence for a comparison result. Statistically, the confidence quantitatively specifies how reliable a proposition is, which is usually a real number in the interval $[0, 1]$. A larger confidence implies that the corresponding proposition is more reliable. In order to show the importance of confidence, here we present an example in the context of performance comparisons of computers. To be specific, the performance speedup of one computer over another is traditionally obtained by comparing the geometric mean performance of one computer (over different benchmarks) with that of another, a practice adopted by SPEC.org [1]. Following this methodology, the performance speedup of PowerEdge T710 over Xserve on SPEC CPU2006, estimated by the data collected from SPEC.org [1], is 3.50. However, after checking this performance speedup with the hierarchical performance testing (HPT) technique suggested in later parts of this paper, we found that the confidence of such a performance speedup is only 0.31, which is rather unreliable (≥ 0.95 is the

- *T. Chen, Y. Wu, Y. Bao, Z. Xu and Y. Chen are with State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China. E-mail: {chentianshi, wuyue, baoyg, zxu, cyj}ict.ac.cn.*
- *Q. Guo is with Department of ECE, Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh, PA 15213 USA. E-mail: qguo1@andrew.cmu.edu.*
- *O. Temam is with Saclay, INRIA, 2-4 rue J. Monod, Orsay, France. E-mail: olivier.temam@inria.fr.*

Manuscript received 20 Aug. 2013; revised 14 Jan. 2014; accepted 5 Mar. 2014. Date of publication 3 Apr. 2014; date of current version 8 Apr. 2015. Recommended for acceptance by R. Gupta.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TC.2014.2315614

statistically acceptable level of confidence). In fact, the HPT technique reports that the reliable performance speedup is 2.24 (with a confidence of 0.95), implying that the conclusion made by comparing geometric mean performance of PowerEdge T710 and Xserve brings an error of 56.2 percent for the quantitative comparison.

The above example shows how confidence plays a critical role in the performance comparison of computers. Nevertheless, we observe that few computer architecture studies yet acknowledge the importance of confidence for performance comparisons and observations: among 521 papers surveyed at ISCA (194 papers, 2006–2010), HPCA (158 papers, 2006–2010) and MICRO (169 papers, 2006–2009), only 28 papers (5.4 percent) resort to confidence estimates in order to assess the variability of performance observations, only 26 (5 percent) rely upon confidence interval, and only 3 (0.57 percent) use t-test. In the meantime, some wrongful practices has also plagued the usage of confidence. For example, parametric statistical techniques like confidence interval and t-test require the sample mean of the performance observations to be distributed normally, which must be guaranteed by either a normal performance distribution or a sufficiently-large number of performance observations. However, such techniques are often incorrectly used when neither of the above preconditions holds. Such practices may potentially bias performance comparisons, and consequently, sometimes leads to incorrect decisions.

In this paper, we provide a novel statistical interpretation to performance comparisons of computers, and review preconditions under which the traditional t-statistics work correctly. Such preconditions are carefully evaluated by experiments and data analysis in the context of computer architecture research, from which we are able to refine guidelines for correctly using these techniques in day-to-day practices of performance comparisons. For scenarios that cannot be appropriately tackled by parametric techniques, we suggest a hierarchical performance testing framework which integrates *non-parametric Statistic Hypothesis Tests* such as Wilcoxon Signed-Rank Test [8], [9]. The most notable merit of the HPT, inherited from non-parametric statistics, is that it works even when there are only a few performance observations which do not obey any specified distribution (e.g., normal distribution). For a qualitative performance comparison, the HPT provides confidence for a proposition like "computer A is faster than computer B". For a quantitative performance comparison, it provides the performance speedup of one computer over another as well as the corresponding confidence. We empirically compare the HPT with a common practice (i.e., using geometric mean performance measure in performance comparisons) in computer architecture research over performance data of commercial computer systems collected from SPEC.org [1]. Compared with the HPT, the common practice which uses geometric mean performance scores to estimate the performance speedup of one computer over another has errors of 8.0 to 56.3 percent on SPEC CPU2006 or SPEC MPI2007.

In summary, the contributions of this paper are the following. First, we empirically highlight that traditional performance comparisons can be unreliable because of the non-deterministic nature of computer performance. As a result, we stress that every performance comparison should come with a confidence estimate in order to judge whether a comparison result corresponds to a stochastic effect or whether it is significant enough. Second, we formulate performance comparison as a statistical task, and investigate in depth why intuitive solutions, t-statistics, are not applicable from a statistical perspective. The investigation is rooted in intensive experiments or publicly-available performance data of commercial computer systems. Third, we propose and implement the HPT framework based on non-parametric statistics, which can provide a sound quantitative estimate of the confidence of a comparison, independently of the distribution and the number of performance observations. This HPT framework has been implemented as an open-source software [10], and integrated in the PARSEC 3.0 benchmark suite [11].

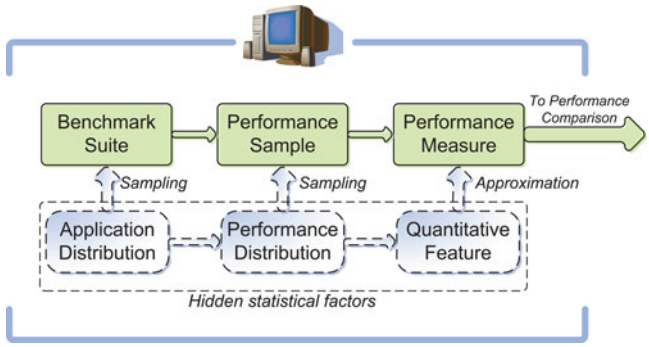

Fig. 1. Statistical performance comparison.

The rest of the paper is organized as follows. Section 2.2 formulates a performance comparison as a statistical task, and reviews the t-statistics. Section 3 empirically checks preconditions of t-statistics, and finds the cases in which t-statistics are not applicable are quite common. Section 4 introduces the non-parametric hierarchical performance testing framework. Section 5 empirically compares the HPT with traditional performance comparison techniques. Section 7 reviews the related work.

2 Performance Comparison as a Statistical Task

In this section, we formulate performance comparisons of computers as a statistical task, and present some basic concepts related to this task. Moreover, we also briefly review the classic t-statistics in the context of performance comparisons (of computers).

2.1 Basic Concepts

From a statistical viewpoint, the non-deterministic performance of a computer is a random variable obeying a certain probability distribution called *performance distribution*. The performance distribution assigns probability/probability density that the performance metric of a computer takes a certain value when encountering a random application coming from another probability distribution called "*application distribution*". The application distribution specifies how likely an application is executed on the computer. Fig. 1 illustrates the relationship between a performance distribution and an application distribution, where the benchmark suite (e.g, SPECint2006 and

SPECfp2006) utilized to evaluate the performance of the computer can be viewed as a representative sample of the application distribution.

When comparing the performance of two computers, we are implicitly comparing the performance distributions of two computers over the same application distribution. In theory, there are different quantitative features to compare two distributions (e.g., distribution mean or distribution median). However, close-form density functions of performance distributions are often unknown, thereby it is hard to analytically deduce such quantitative features. Fortunately, the above task can be accomplished by *statistical sampling*, a process that collects a number of performance observations[1] of the performance distribution. The performance observations make up a *performance sample* of the computer, which can be used to estimate the *performance measure*. In practical performance comparisons, the performance measure (e.g., sample mean) is an indicator of computer performance, which, in our viewpoint, is an approximation to the corresponding quantitative feature (e.g., distribution mean) of the performance distribution. Fig. 1 depicts the whole statistical task, where upper blocks represent concepts visible to practitioners, and lower blocks represent hidden but underlying statistical factors which support the statistical task.

2.2 A Review to *t*-Statistics

According to the law of large numbers, if we have an infinite number of performance observations for each computer, then the performance distribution of each computer as well as its quantitative features can be accurately captured, and the performance comparison would become straightforward and accurate. However, limited by the number of performance observations that we collect in practice, performance sampling will always bring in stochastic error. Under this circumstance, it is necessary to introduce a quantitative indicator called *confidence* to judge whether a comparison result corresponds to a stochastic effect or whether it is significant enough to accept.

In practice, the scopes of most commercial computers have never been restricted to specific benchmark applications only, thereby a performance comparison result obtained on benchmarks shall be generalizable to broader applications other than benchmarks. From a practical (but not rigorous) viewpoint, the confidence indicates how likely the comparison result can be generalized to a new application[2]. Hence, it is important not only to assess the confidence of a performance comparison, but also to correctly evaluate this confidence.

Traditionally, t-statistics have long been considered to be powerful in estimating confidences of performance evaluations of computers [12], [13]. Belonging to parametric statistics, t-statistics rely on an assumption that data directly come from a specific type of probability distributions called normal distributions, or can be characterized by normal distributions after certain data transformations. t-statistics enable two famous statistical inference techniques called t-test and t-distribution confidence interval (t-confidence interval for short). This section briefly reviews the principle of t-statistics, as well as preconditions under which t-statistics correctly work.

Consider a sample $\{X_1, \ldots, X_n\}$ with n observations of the same population distribution with finite mean and variance. The sample mean $\overline{X}$ and sample standard deviation S are defined by:

$$\overline{X} = \frac{1}{n}(X_1 +, \ldots, +X_n), \tag{1}$$

$$S = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(X_i - \overline{X})^2}. \tag{2}$$

Statistically, *if $\overline{X}$ obeys a normal distribution*, then

$$T = \frac{\sqrt{n}(\overline{X} - \mu)}{S} \tag{3}$$

obeys the student's t-distribution with $n-1$ degrees of freedom, where μ is the mean of the population distribution.

The famous paired two-sample t-test is based on the above fact, which directly tells us "one is significantly larger than another" or "there is no significant difference between the two". It was thought to be useful in performance comparisons of computers [13], because it compares the corresponding performance of two computers on each benchmark, and summarizes performance gaps on different benchmarks to statistically compare the mean performance of two computers. Consider two ordered samples $\{A_1, \ldots, A_n\}$ and $\{B_1, \ldots, B_n\}$ obeying two population distributions $\mathcal{A}$ and $\mathcal{B}$, respectively. Let D_i $(i = 1, \ldots, n)$ be the difference between A_i and B_i, $\overline{D}$ and S_D be the sample mean and sample standard deviation of $\{D_1, \ldots, D_n\}$, respectively. If $\overline{D}$ obeys a normal distribution, then

$$T = \frac{\sqrt{n}\,\overline{D}}{S_D} \tag{4}$$

obeys the student's t-distribution with $n-1$ degrees of freedom. Eq. (4) can be viewed as a variant of Eq. (3), where $\overline{X}$ in Eq. (3) is replaced by $\overline{D}$ here. With the above t-statistics, paired two-sample t-test statistically compares population means of distributions $\mathcal{A}$ and $\mathcal{B}$. In the context of performance comparisons of computers, *paired two-sample t-test requires that the sample mean of performance gaps between two computers, $\overline{D}$, must obey a normal distribution in order to apply t-statistics*. In practice, normally distributed $\overline{D}$ can be achieved with either small-sample or large-sample preconditions:

- *Small-sample precondition*. Performance distributions of both computers are normal;
- *Large-sample precondition*. When performance distributions of one or both computers are non-normal but are with finite means and variances, the Central Limit Theorem (CLT) states that the sample mean performance of both computers approximately obeys normal distributions when

1. A performance observation of a computer is the performance score of the computer measured in a run of an application.

2. This is the case of cross-application comparison, where we care about the performance of each computer on multiple applications. If talking about a uni-application comparison, the confidence indicates how likely the comparison result obtained on existing runs of the same application can be generalized to more runs of the same application.

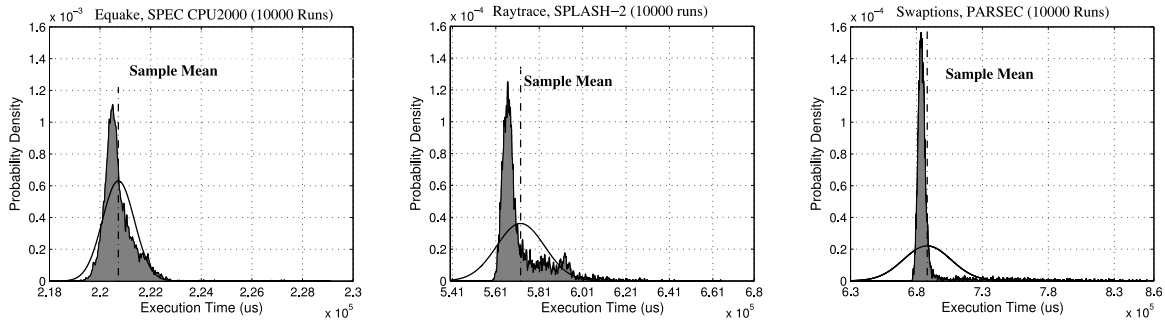

Fig. 2. Estimating probability density functions (PDFs) on *Equake* (SPEC CPU2000), *Raytrace* (SPLASH-2) and *Swaptions* (PARSEC) by KPW (black curves above the grey areas) and NNF (black curves above the white areas), from 10,000 repeated runs of each benchmark on the same computer.

the sample size n (number of performance observations) is *sufficiently large*.

In the next section, we empirically study the above preconditions in the context of computer performance.

3 Empirical Observations

In this section, we empirically study whether small-sample and large-sample preconditions of t-statistics hold under different scenarios of computer architecture research. The central claim of this section is that t-statistics cannot be the de facto solution for performance comparisons of computers.

3.1 Small-Sample Precondition

The small-sample precondition of t-statistics states that the performance distribution of computer-under-comparison should be normal. In this part, we check this precondition under both uni-application and cross-application comparison scenarios.

3.1.1 Uni-Application Performance Distribution

The performance of a computer on every application is influenced by not only architecture factors (e.g., out-of-order execution, branch prediction, and chip multiprocessor [14]) but also program factors (e.g., data race, synchronization, and contention of shared resources [15], [16]). In the presence of these factors, the performance score of a computer on a same application is usually non-deterministic [4]. For example, according to our experiments using SPLASH-2 [2], the execution time of one run of a benchmark can be up to 1.27 times that of another run of the same benchmark on the same computer. Recent studies managed to reduce performance variation and simultaneously improve the performance [17], or enforce computer performance to be distributed normally [18]. Yet it is still interesting to empirically study the real probability distribution of non-deterministic computer performance, with which we may select appropriate statistical techniques to facilitate the performance evaluation.

In our experiments, we run both single-threaded (*Equake*, SPEC CPU2000 [1]) and multi-threaded benchmarks (*Raytrace*, SPLASH-2 [2] and *Swaptions*, PARSEC [3]) on a commodity Linux workstation with a 4-core 8-thread CPU (Intel i7 920) and 6 GB DDR2 RAM. Each benchmark is repeatedly run for 10,000 times, respectively. In each run, *Equake* uses the "test" input defined by SPEC CPU2000, *Raytrace* uses the largest input given by SPLASH-2 (car.env), and *Swaptions* uses the second largest input of PARSEC (simlarge). Without losing any generality, we define the performance score to be the execution time.

To study *uni-application* performance distributions of the computer on different benchmarks, we employ a statistical technique called Kernel Parzen Window (KPW) [19]. The KPW technique estimates a performance distribution without assuming that the performance distribution is normal or some specific distribution. Instead, it studies the real performance distribution in a Monte-Carlo style, and directly estimates the probability density function via histogram construction and Gaussian kernel smoothing. As the reference, we also employ a simple technique called naive normality fitting (NNF) to process the same performance data. The NNF always assumes that each performance distribution is normal, and directly uses the mean and standard deviation of a performance sample (with multiple performance observations) as the mean and standard deviation of the normal performance distribution. By comparing PDFs obtained by two techniques, we can easily identify whether or not a performance distribution obeys a normal law. Specifically, if the normal distribution obtained by the NNF complies with the real performance distribution estimated by the KPW, then the performance distribution obeys a normal law. Otherwise, it does not.

According to the experimental results illustrated in Fig. 2, the normality does not hold for the performance score of the computer on all three benchmarks, as evidenced by the remarkably long right tails and short left tails of the estimated performance distributions for *Equake*, *Raytrace* and *Swaptions*. Such observations are surprising but not counter-intuitive due to the intrinsic non-determinism of computers and applications. In short, it is hard for a program to execute faster than a threshold, but easy to be slowed down by various events, especially for multi-threaded programs which are affected by data races, thread scheduling, synchronization order, and contentions of shared resources.

As a follow-up experiment, we use a more rigorous statistical technique to study whether execution times of the 27 benchmarks of SPLASH-2 and PARSEC (using "simlarge" inputs) are distributed normally, where each benchmark is repeatedly run on the commodity computer for 10000 times again. Based on these observations, the Lilliefors test (Kolmogorov-Smirnov test) [20] is utilized to

TABLE 1
SPEC Ratios of BL265+ on SPECint2006 (Top) and SPECfp2006 (Bottom) [1]

SPECint2006	perlbench	bzip2	gcc	mcf	gobmk	hmmer
SPEC Ratio	25.9	19.5	26.8	50.4	23.9	47.6

SPECint2006	sjeng	libquantum	h264ref	omnetpp	astar	xalancbmk
SPEC Ratio	26.6	992.8	37.8	23.6	24.3	39.3

SPECfp2006	bwaves	gamess	milc	zeusmp	gromacs	cactusADM	leslie3d	namd	dealII
SPEC Ratio	174.5	23.5	52.7	113.1	21.9	279.6	110.3	19.5	40.5

SPECfp2006	soplex	povray	calculix	GemsFDTD	tonto	lbm	wrf	sphinx3
SPEC Ratio	33.2	30.3	29.8	73.1	25.0	262.0	49.1	53.1

estimate the confidence that the execution time does not obey the normal law, i.e., the confidence that the normality assumption is incorrect. Interestingly, it is observed that for *every* benchmark of SPLASH-2 and PARSEC, the confidence that the normality assumption is incorrect is above 0.95. Our observation with SPLASH-2 and PARSEC is significantly different from the observation of Georges et al. [21] that uni-application performance on single cores (using SPECjvm98) is distributed normally, suggesting that the non-deterministic performance of multi-threaded programs is fairly different from that of single-threaded programs.

3.1.2 Cross-Application Performance Distribution

After studying performance distributions of computers on single applications, we now consider performance distributions over multiple applications, which are called *cross-application* performance distributions here. Traditionally, there have been assumptions that performance distributions are normal [12] or log-normal [22]. In this part, we empirically evaluate such assumptions. Table 1 presents performance scores (SPEC ratios) of a commodity computer (BL265+, Intel Xeon X5670, 2.93 GHz) over SPEC CPU2006, where the data is collected from the SPEC online repository [1]. It can be observed that the SPEC ratios of the computer on most benchmarks are between 19 and 40. However, the SPEC ratios on a few benchmarks (e.g., *libquantum* and *cactusADM*) are remarkably high, which prevent the cross-application performance distributions from being distributed normally. Fig. 3 validates that the cross-application performance distributions of the computer are non-normal using the normal probability plot [23]. In each probability plot presented in Fig. 3, if the curve matches well the straight line, then the performance distribution over the corresponding benchmark suite is normal; if the curve departs from the straight line, then the performance distribution is not normal. Obviously, none of the figures shows a good match between the curve and straight line, implying that performance distributions of the computer over SPEC CPU2006, SPECint2006 and SPECfp2006 are not normal. As another piece of statistical evidence, Lilliefors test [20] concludes that performance distributions of the computer over SPEC CPU2006, SPECint2006 and SPECfp2006 are non-normal with a confidence larger than 0.95. The above data analysis shows that the normality of performance distributions is vulnerable to positive performance outliers (e.g., the remarkably high SPEC CPU2006 ratio of BL265+ on *libquantum*). Statistically, the existence of positive performance outliers implies that the corresponding performance distribution is skewed and long-tailed, while a normal distribution should be symmetric.

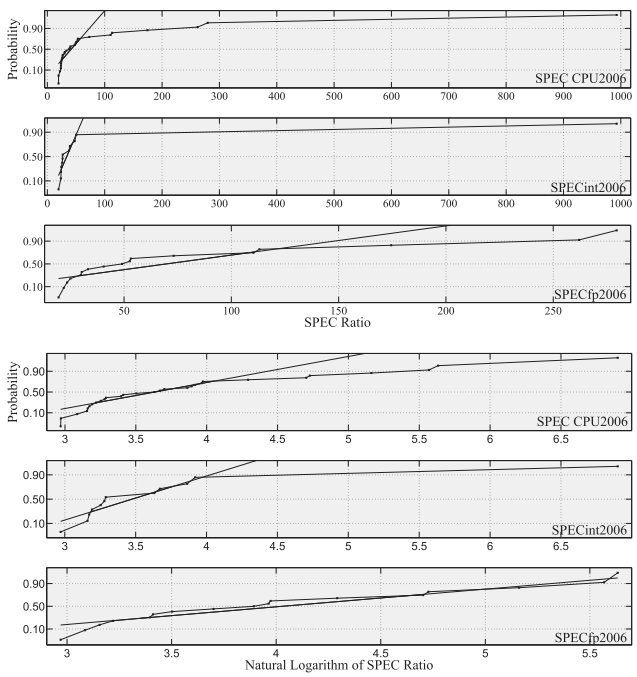

Fig. 3. Graphically assessing whether performance distributions of a commodity computer are normal (normal probability plots [23], left figures) or log-normal (log-normal probability plots, right figures) using natural logarithms of SPEC ratios of SPEC CPU2006, SPECint2006 and SPECfp2006.

Instead of directly assuming the normality of performance distributions, Mashey suggested an alternative assumption that cross-application performance distributions are log-normal [22], i.e., performance observations are distributed log-normally.[3] Although a logarithm transformation can alleviate the impact of performance outliers as well as the risk of using t-statistics, when the performance of very few outliers deviates too much from the typical performance of a computer, the log-normality may still fail to appropriately characterize the performance outlier. Hereinafter we still take the commodity computer (BL265+, Intel Xeon X5670, 2.93 GHz) as an example, and analyze whether its SPEC ratio is distributed log-normally. According to Fig. 3, the performance distribution over SPECfp2006 is almost log-normal (the curve almost matches the straight line), but distributions over SPEC CPU2006 and SPECint2006 are apparently not log-normal.

3. A random variable a is said to obey a log-normal distribution if its logarithm $\ln a$ is distributed normally.

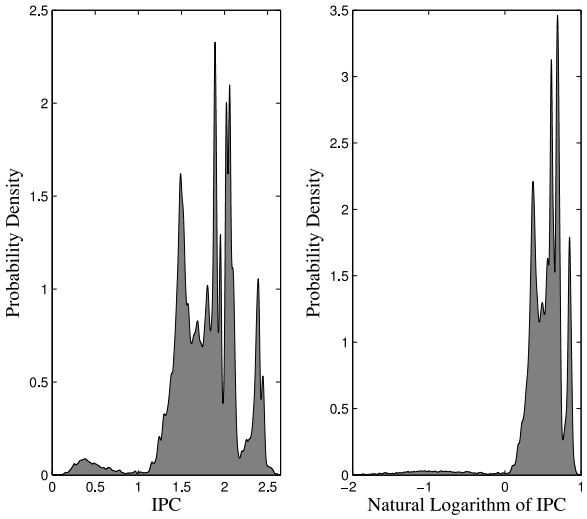

Fig. 4. Probability density functions of the performance (left) and log-performance (right) of Intel Xeon dualcore Linux workstation, estimated from KDataSets. The performance of the computer is measured by the instruction-per-cycle (IPC).

That is because after a logarithm transformation the SPEC ratio of the computer on the SPECint2006 benchmark *libquantum* is still too large to be characterized by a normal distribution.

In summary, performance outliers have already been quite common in performance reports of latest commodity computers, and have been significant enough to break the normality/log-normality of computer performance. SPEC CPU2006 reports of most (if not all) commodity computers published in 2012 at SPEC.org [1] clearly confirm existence of performance outliers. In the era of dark silicon, this trend will become even more distinct, as specialized hardware accelerators designed for specific applications may produce more significant performance outliers. Under this circumstance, the normality cannot be a default assumption for performance distributions of computers.

3.2 Large-Sample Precondition

So far we have empirically shown that the performance distributions of computers cannot be universally characterized by normal distributions. According to the large-sample precondition, however, it is still possible to obtain a normally distributed sample mean of performance observations (in order to apply t-statistics) given a sufficiently large number of observations, as guaranteed by the Central Limit Theorem. The classical version of the CLT contributed by Lindeberg and Lévy [24] states that, when the sample size n is sufficiently large, the sample mean approximately obeys a normal distribution. Nevertheless, in practice, it is unclear how large the sample size should be to address the requirement of "a sufficiently large sample". In this part, we empirically show that the appropriate sample size for applying the CLT is usually too large to be compatible with current practices in computer performance observations.

In order to obtain a large number of performance observations, KDataSets [25] is used in our experiments. A notable feature of KDataSets is that it provides 1,000 distinct data sets for each of 32 different benchmarks (MiBench [26]), providing a total of 32,000 distinct runs. We collect detailed performance scores (measured in instruction-per-cycle, IPC) of a Linux workstation (with 3 GHz Intel Xeon dualcore processor, 2 GB RAM) over 32,000 different combinations of benchmarks and data sets of KDataSets [25]. With the 32,000 performance scores, we estimate probability distributions of IPC and IPC's natural logarithm, which are illustrated in Fig. 4. Clearly, both IPC and IPC's natural logarithm are not distributed normally, which well complies our discussions in previous sections.

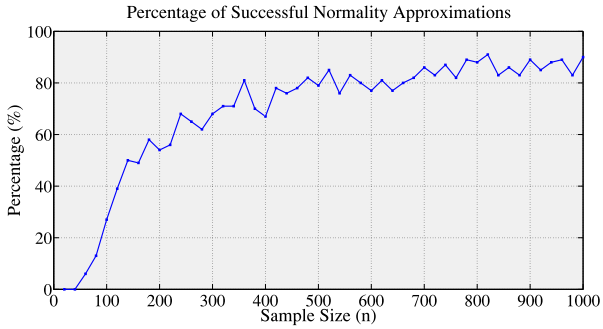

Fig. 5. Percentage of successful normality approximation with respect to the sample size.

Now we study in detail what is a sufficient sample size for applying the CLT on the above performance distribution. To be specific, for every fixed sample size $n \in \{20, 40, 80, \ldots, 980, 1{,}000\}$, we conduct 100 trials and estimate the percentage of successful normality approximations among the 100 trials. In each trial, we collect $10{,}000$ samples, each of which is comprised of n performance scores randomly selected out of the 32,000 performance scores (with replacement). As a consequence, we get $10{,}000$ observations of sample mean, which are sufficient to reliably identify whether the sample mean is distributed normally. After that, we use a normality test (Lilliefors test) to detect whether the sample mean is distributed normally at the significance level of 0.05 (i.e., whether the normality approximation is successful).

Fig. 5 depicts percentages of successful normality approximations with respect to different sample sizes. Clearly, the sample mean is not distributed normally given a small (e.g., $n = 20, 40$) sample size. When the sample size grows to 100, the chance that the sample mean is not distributed normally is above 70 percent. Even when the sample size grows up to 500, the sample mean still takes a 21 percent chance to depart from the normality. The above observations imply that a sample with several hundreds of performance observations is necessary to get a normally distributed sample (arithmetic) mean, at least for the studied computer on this benchmark suite.

In addition, we also explore the sufficient sample size in order to approximate a log-normally distributed sample mean. The experimental setting is the same to the one introduced above, except that performance data of KDataSets are pre-processed by a logarithm transformation. We illustrate the percentage of successful log-normality approximations in Fig. 6. Compared with the percentage of successful normality approximations illustrated in Fig. 5, the percentage of successful log-normality approximations is much smaller under the same sample size. Surprisingly,

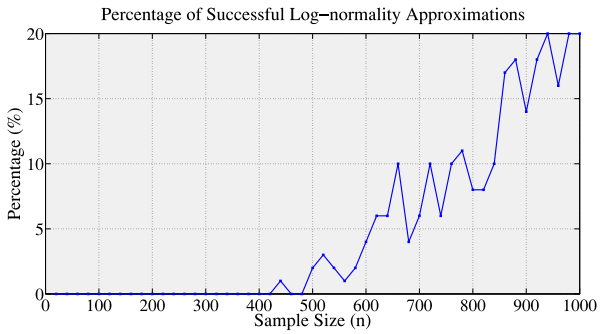

Fig. 6. Percentage of successful log-normality approximation with respect to the sample size.

the log-normality approximations cannot be successful when the sample size is smaller than 420. Even when the sample size is 1,000, the percentage of successful log-normality approximations is only 20 percent. An explanation to such observations is that the logarithm transformation of data results in a distribution with larger negative skew (refer to Fig. 4 for the log-performance distribution with the long left tail),[4] which even differs more from a normal distribution (compared with the original performance distribution). Applying the CLT on such a skewed distribution may have to require a large sample size.

The common insight gained from the above experiments is that the number of performance observations (i.e., sample size) for approximating normal/log-normal distributions with the CLT is very large (e.g., several hundreds), thus can rarely be collected in day-to-day practices using only 20-30 benchmarks (with one to a few data sets each).

3.3 A Practical Example

We have empirically studied several fundamental statistical issues closely related to the effectiveness of parametric techniques like t-test, and the key observation is that preconditions (normal performance distributions or sufficiently large performance samples) required by such techniques may not hold in computer architecture research. Here we offer an example showing how t-test fails to present a reasonable comparison result when neither of its preconditions holds.

In this example, we compare the performance of BL265+ (Intel Xeon X5670, 2.93 GHz) and CELSIUS R550 (Intel Xeon E5440, 2.83 Hz) over SPECint2006. The SPEC ratios of the computers, collected from SPEC.org [1], are presented in Table 2, respectively. Clearly, the performance distributions of both computers are non-normal due to the outlier performance on *libquantum*, and sizes of both performance samples are only 12 (far from enough for applying the CLT). In other words, preconditions required by paired t-test do not hold. Voluntarily ignoring that fact, we incorrectly use the paired t-test, and get the conclusion "BL265+ does not significantly outperform CELSIUS R550 at the significance level 0.05 (the confidence level 0.95)". This conclusion apparently contradicts the straightforward fact that BL265+ outperforms CELSIUS R550 on *all* 12 benchmarks of SPECint2006. A detailed explanation to the failure of t-test is as following. In this example, t-test does a brute-force normality fitting[5] to a right-skewed non-normal performance distribution, and incorrectly stretches the left part of the right-skewed distribution to achieve a symmetric distribution. In this process, the large standard deviation originally contributed by the *libquantum* performance outlier is thought to be symmetrically distributed at both left and right sides, and the left tail is elongated, which produces an illusion that the computer even takes a high probability to have negative SPEC ratio. Such absurd arguments eventually lead to the incorrect comparison result. In summary, it is important to correctly use statistical techniques in performance comparisons.

4 Non-Parametric Hierarchical Performance Testing Framework

One crucial branch of statistical inference is called statistical hypothesis test (SHT). Generally speaking, an SHT is a procedure that makes choices between two opposite hypotheses (propositions), NULL (default) hypothesis and alternative hypothesis. The NULL hypothesis represents the default belief, i.e., our belief before observing any evidence, and the alternative hypothesis (often the claim we want to make) is a belief opposite to the NULL hypothesis. In performance comparison, a typical NULL hypothesis may be "computer A is as fast as computer B", and a typical alternative hypothesis may be "computer A is faster than computer B". At the beginning of an SHT, one assumes the NULL hypothesis to be correct, and constructs a statistic (say, Z) whose value can be calculated from the observed data. The value of Z determines the possibility of observing the current data when assuming the NULL hypothesis holds, which is critical for making a choice between the NULL hypothesis and the alternative hypothesis. The possibility is quantified as $p\text{-}value$ (or significance probability) [27], which is a real value between 0 and 1 that can simply be considered as a measure of *risk* associated with the *alternative* hypothesis. The $p\text{-}value$ is an indicator for decision-making: when the $p\text{-}value$ is small enough, the risk of incorrectly rejecting the NULL hypothesis is very small, and the confidence of the alternative hypothesis (i.e., $1 - p\text{-}value$) is large enough. For example, in an SHT, when the $p\text{-}value$ of the NULL hypothesis "computer A is as fast as computer B" is 0.048, we only have a 4.8 percent chance of rejecting the NULL hypothesis when it actually holds. In other words, the alternative hypothesis "computer A is faster than computer B" has confidence $1 - 0.048 = 0.952$. Closely related to the $p\text{-}value$, significance levels act as scales of a ruler for the $p\text{-}value$ (frequently-used scales include 0.001, 0.01, 0.05, and 0.1). A significance level $\alpha \in [0, 1]$, can simply be viewed as the confidence level $1 - \alpha$. As a statistical convention, a confidence no smaller than 0.95 is often necessary for reaching the final conclusion.

Probably the most well-known parametric SHTs are the family of t-tests. Rather than sticking to the well-known t-test, we introduce a hierarchical performance testing

4. The logarithm transformation can be utilized to reduce the impact of outliers having extremely *large* values, but may even enhance the impact of outliers having very *small* values.

5. When the sample size is small (<30), fitting a normal distribution is achieved by fitting a t-distribution.

TABLE 2
SPEC Ratios of BL265+ and CELSIUS R550 on SPECint2006 [1]

SPECint2006	perlbench	bzip2	gcc	mcf	gobmk	hmmer
BL265+	25.9	19.5	26.8	50.4	23.9	47.6
CELSIUS R550	18.5	15.4	14.1	21.5	18.6	14.9
SPECint2006	sjeng	libquantum	h264ref	omnetpp	astar	xalancbmk
BL265+	26.6	992.8	37.8	23.6	24.3	39.3
CELSIUS R550	16.9	212.9	29.3	15.0	14.3	24.8

framework, which does not dictate for small-sample or large-sample precondition as t-test.

4.1 General Flow

In computer architecture research, we often need to compare the performance of two computers on a number of benchmark applications, where each application is run for multiple times on a computer. Designed for such comparison tasks, the HPT combines the performance comparisons at both uni-application and cross-application levels. For each single application, the HPT employs Wilcoxon *Rank-Sum* Test [9] to check whether the performance difference of two computers on that application is significant enough (i.e., the corresponding significance level is small enough), in other words, whether the observed superiority of one computer over another is reliable enough. Only significant (reliable) differences, identified by the SHTs in uni-application comparisons, can be taken into account by the comparison over different benchmarks, while those insignificant differences will be ignored (i.e., the insignificant differences are set to 0) in the comparison over different benchmarks. Based on uni-application performance observations, the Wilcoxon *Signed-Rank* Test [8], [9] is employed to statistically compare the cross-application performance of two computers. Through these non-parametric SHTs, the HPT can quantify confidences of performance comparisons. In this section, the technical details of the HPT will be introduced.[6]

Let us assume that we are comparing two computers A and B over a benchmark suite consisting of n benchmark applications. Each computer repeatedly runs each application m times (SPEC.org sets $m = 3$ [1]). Let the performance scores of A and B at their jth runs on the ith benchmark be $a_{i,j}$ and $b_{i,j}$ respectively. Then the performance samples of the computers can be represented by *performance matrices* $S_A = [a_{i,j}]_{n\times m}$ and $S_B = [b_{i,j}]_{n\times m}$, respectively. For the corresponding rows of S_A and S_B (e.g., the τ-th rows of the matrices, $\tau = 1, \ldots, n$), we carry out the Wilcoxon Rank-Sum Test to investigate whether the difference between the performance scores of A and B is significant enough. The concrete steps of Wilcoxon Rank-Sum Test are the following:

- Let the NULL hypothesis of the SHT be "$H_{\tau,0}$*: the performance scores of A and B on the τth benchmark are equivalent to each other*"; let the alternative hypothesis of the SHT be "$H_{\tau,1}$*: the performance score of A is higher than that of B on the τth benchmark*" or "$H_{\tau,2}$*: the performance score of B is higher than that of A on the τth benchmark*", depending on the motivation of carrying out the SHT. Define the significance level be α_τ; we suggest setting $\alpha_\tau = 0.05$ for $m \geq 5$ and 0.10 for the rest cases.
- Sort $a_{\tau,1}, a_{\tau,2}, \ldots, a_{\tau,m}, b_{\tau,1}, b_{\tau,2}, \ldots, b_{\tau,m}$ in ascending order, and assign each of the scores the corresponding rank (from 1 to $2m$). In case two or more scores are the same, but their original ranks are different, renew the ranks by assigning them the average of their original ranks.[7] Afterwards, for A and B, their rank sums (on the τth benchmark) can be defined as:

$$\mathrm{R}_{a,\tau} = \sum_{j=1}^{m} \mathrm{Rank}_\tau(a_{\tau,j}), \quad \mathrm{R}_{b,\tau} = \sum_{j=1}^{m} \mathrm{Rank}_\tau(b_{\tau,j}),$$

  where $\mathrm{Rank}_\tau(\cdot)$ provides the rank of a performance score on the τ-th benchmark.
- *Case [$m < 12$]*[8]. When the alternative hypothesis of the SHT is $H_{\tau,1}$, we reject the NULL hypothesis and accept $H_{\tau,1}$ if $\mathrm{R}_{a,\tau}$ is *no smaller than* the critical value (right tail, Wilcoxon Rank-Sum Test) under the significance level α_τ. When the alternative hypothesis of the SHT is $H_{\tau,2}$, we reject the NULL hypothesis and accept $H_{\tau,2}$ if $\mathrm{R}_{b,\tau}$ is *no smaller than* the critical value under the significance level α [28].
- *Case [$m \geq 12$]*. Define two new statistics $z_{a,\tau}$ and $z_{b,\tau}$ as follows:

$$z_{a,\tau} = \frac{\mathrm{R}_{a,\tau} - \frac{1}{2}m(2m+1)}{\sqrt{\frac{1}{12}m^2(2m+1)}}, \quad z_{b,\tau} = \frac{\mathrm{R}_{b,\tau} - \frac{1}{2}m(2m+1)}{\sqrt{\frac{1}{12}m^2(2m+1)}}.$$

  Under the NULL hypothesis, $z_{a,\tau}$ and $z_{b,\tau}$ approximately obey the standard normal distribution $\mathcal{N}(0,1)$. When the alternative hypothesis of the SHT is $H_{\tau,1}$, we reject the NULL hypothesis and accept $H_{\tau,1}$ if $z_{a,\tau}$ is *no smaller than* the critical value (right tail, standard normal distribution) under the significance level α; when the alternative hypothesis of the SHT is $H_{\tau,2}$, we reject the NULL hypothesis and accept $H_{\tau,2}$ if $z_{b,\tau}$ is *no smaller than* the critical value under the significance level α [28].

  After carrying out the above SHT with respect to the τ-th benchmark ($\tau = 1, \ldots, n$), we are able to assign the difference (denoted by d_τ) between the performance of A and B. Concretely, if the SHT accepts $H_{\tau,1}$ or $H_{\tau,2}$ with a promising significance level (e.g., 0.01 or 0.05), then we let

6. Users who are not interested in mathematical details of the non-parametric SHTs can omit the rest of this section.

7. For example, if two scores are both 50, and their original ranks are 5 and 6 respectively, then both of them obtain a rank of 5.5.

8. In statistics, when $m < 12$, the critical values for Wilcoxon rank sum test are calculated directly. When $m \geq 12$, the corresponding critical values are often estimated by studying the approximate distribution of the rank sum.

$$d_\tau = \text{median}\{a_{\tau,1}, a_{\tau,2}, \ldots, a_{\tau,m}\} - \text{median}\{b_{\tau,1}, b_{\tau,2}, \ldots, b_{\tau,m}\}, \quad \tau = 1, \ldots, n.$$

Otherwise (if the NULL hypothesis $H_{\tau,0}$ has not been rejected at a promising significant level), we let $d_\tau = 0$, i.e., we ignore the insignificant difference between the performance scores of A and B. $d_1, d_2, \ldots, d_n$ will then be utilized in the following Wilcoxon Signed-Rank Test for the performance comparison over different benchmarks:

- Let the NULL hypothesis of the SHT be "H_0: *the general performance of A is equivalent to that of B*"; let the alternative hypothesis of the SHT be "H_1: *the general performance of A is better than that of B*" or "H_2: *the general performance of B is better than that of A*", depending on the motivation of carrying out the SHT.
- Rank $d_1, d_2, \ldots, d_n$ according to an ascending order of their absolute values. In case two or more absolute values are the same, renew the ranks by assigning them the average of their original ranks. Afterwards, for A and B, their signed-rank sums can be defined as:

$$\mathrm{R}_A = \sum_{i:d_i>0} \text{Rank}(d_i) + \frac{1}{2}\sum_{i:d_i=0} \text{Rank}(d_i),$$
$$\mathrm{R}_B = \sum_{i:d_i<0} \text{Rank}(d_i) + \frac{1}{2}\sum_{i:d_i=0} \text{Rank}(d_i),$$

  where $\text{Rank}(d_i)$ provides the rank of the absolute value of d_i, which was described above.
- *Case [$n < 25$]*[9]. When the alternative hypothesis of the SHT is H_1, we reject the NULL hypothesis and accept H_1 if R_B is *no larger than* the critical value (one-side Wilcoxon Signed-Rank Test) under the significance level α; When the alternative hypothesis of the SHT is H_2, we reject the NULL hypothesis and accept H_2 if R_A is *no larger than* the critical value under the significance level α. The critical values of Wilcoxon Signed-Rank Test are available in statistics books [28].
- *Case [$n \geq 25$]*. Define two new statistics z_A and z_B as

$$z_A = \frac{\mathrm{R}_A - \frac{1}{4}n(n+1)}{\sqrt{\frac{1}{24}n(n+1)(2n+1)}}, \; z_B = \frac{\mathrm{R}_B - \frac{1}{4}n(n+1)}{\sqrt{\frac{1}{24}n(n+1)(2n+1)}}.$$

Under the NULL hypothesis, z_A and z_B approximately obey the standard normal distribution $\mathcal{N}(0,1)$. Hence, when the alternative hypothesis of the SHT is H_1, we reject the NULL hypothesis and accept H_1 if z_B is *no larger than* the critical value (lower tail, standard normal distribution) under the significance level α; when the alternative hypothesis of the SHT is H_2, we reject the NULL hypothesis and accept H_2 if z_A is *no larger than* the critical value under the significance level α. For the comparison over different benchmarks, the outputs of the HPT, including the comparison result and its confidence, are finally presented by the above Wilcoxon Signed-Rank Test. Formally, given a fixed significance level α for the HPT, we utilize $Confidence(\text{HPT} : S_A \succ S_B) \geq r$ to represent the following conclusion made by the HPT: "A outperforms B with the confidence r", where $r = 1 - \alpha$.

9. In statistics, when $n < 25$, the critical values for Wilcoxon signed rank test are calculated directly. When $n \geq 25$, the corresponding critical values are often estimated by studying the approximate distribution of the signed rank sum.

4.2 Statistical Performance Speedup Testing

So far we have shown how to carry out qualitative performance comparisons with the HPT. In addition to qualitative comparisons, in most cases we are more interested in quantitative comparison results such as "Computer A is more than γ times faster than Computer B", where $\gamma \geq 1$ is defined as the *speedup-under-test*. Traditionally, such kind of arguments are often obtained directly by comparing the means of performance scores with respect to computers A and B. Taking the SPEC convention as an example, if the mean (geometric) SPEC ratios of A is ten times that of B, then one would probably conclude that "A is ten times faster than B". Such a quantitative comparison is dangerous since we do not know how much we can trust the result. Fortunately, the HPT framework offers two solutions for tackling speedup arguments. The first solution requires us to specify the concrete value of γ before the test. Afterwards, we shrink performance scores of computer A by transforming the corresponding performance matrix S_A to S_A/γ (without losing generality, we employ normalized performance ratio as performance score with respect to each benchmark, where a larger performance score means better performance). Considering a virtual computer with performance matrix S_A/γ, if the HPT framework states that the virtual computer outperforms computer B with a confidence r, then we claim "A is more than γ times faster than B with a confidence r". *In general, if we specify a more (less) conservative speedup γ before speedup testing, the corresponding speedup argument will have a larger (smaller) confidence r.* Users should keep a balance between the speedup and the corresponding confidence, so as to make a convincing yet not-too-conservative conclusion.

In many cases, instead of deciding a speedup γ before the statistical test, one would like to know *the largest speedup that results in a reliable comparison result* (for a given confidence r). To address this need, our HPT framework also offers an alternative way of estimating the speedup and corresponding confidence. Guided by the above notion, we formally define the r-Speedup (computer A over computer B, r-$\text{Speedup}(A, B)$) to be

$$\sup\left\{\gamma \geq 1; confidence\left(\text{HPT} : \frac{1}{\gamma}S_A \succ S_B\right) \geq r\right\}.$$

To be specific, the r-Speedup of computer A over computer B is the largest speedup of A over B having a confidence above r. The algorithm of computing the r-Speedup is presented in Algorithm 1, where the precision is set to 2 decimals, and $HPT(X, Y)$ presents the confidence that the computer w.r.t. the performance sample X outperforms the computer w.r.t. the performance sample Y. Given the confidence level r (e.g., $r = 0.95$), the r-Speedup can be viewed as a quantitative indicator of performance speedup with the guarantee of confidence r. Meanwhile,

TABLE 3
Statistical Quantitative Comparison of Computers X and Y over SPLASH-2 (Speedup: 1.76)

	$S_{\tilde{X}}$					Med.	Stat. Win.	Diff. (d_τ)	Rank	Med.	S_Y					
1.barnes	0.53	0.54	0.54	0.53	0.54	0.54	Y	-0.50	10	1.04	1.00	1.05	1.04	1.03	1.04
2.cholesky	0.97	0.95	0.93	0.96	0.96	0.96	Y	-0.03	3	0.99	1.00	0.98	1.01	0.99	0.98
3.fft	0.74	0.76	0.74	0.78	0.76	0.76	Y	-0.27	6.5	1.03	1.00	1.03	1.02	1.05	1.03
4.fmm	1.07	1.03	1.05	1.02	1.05	1.05	Tie	0(0.01)	1.5	1.04	1.00	1.05	1.04	1.04	1.05
5.lu-con	1.29	1.26	1.27	1.27	1.25	1.27	$\tilde{X}$	0.27	6.5	1.00	1.00	1.01	1.02	0.98	1.00
6.lu-ucon	1.46	1.48	1.38	1.53	1.55	1.48	$\tilde{X}$	0.49	9	0.99	1.00	0.96	1.04	0.87	0.99
7.ocean-con	1.17	1.15	0.94	1.16	1.13	1.15	$\tilde{X}$	0.17	5	0.98	1.00	0.91	1.00	0.98	0.86
8.ocean-ucon	1.95	1.98	1.92	1.93	1.93	1.93	$\tilde{X}$	0.95	13	0.98	1.00	0.98	0.97	0.90	0.98
9.radiosity	1.01	1.01	1.01	0.99	1.01	1.01	Tie	0(0.01)	1.5	1.00	1.00	1.00	1.00	1.00	1.00
10.radix	2.47	2.51	2.53	2.44	2.11	2.47	$\tilde{X}$	1.50	14	0.97	1.00	0.86	0.95	1.03	0.97
11.raytrace	1.41	1.39	1.43	1.21	1.37	1.39	$\tilde{X}$	0.32	8	1.07	1.00	1.09	1.07	1.14	1.07
12.volrend	0.92	0.94	0.92	0.92	0.93	0.92	Y	-0.08	4	1.00	1.00	1.00	1.00	1.00	1.00
13.water-ns	1.64	1.66	1.59	1.64	1.63	1.64	$\tilde{X}$	0.69	11	0.95	1.00	0.95	0.84	0.93	0.96
14.water-sp	1.84	1.88	1.78	1.80	1.77	1.80	$\tilde{X}$	0.80	12	1.00	1.00	1.02	0.98	0.87	1.04

it can also act as a single-number performance metric as the geometric mean, which will be discussed in Section 6.

We contribute two implementations of the HPT, including an open-source software [10] and a toolkit in the latest version of PARSEC benchmark suite (PARSEC 3.0 [11]). Users can easily use them without acquiring the mathematical details.

Algorithm 1: r-speedup testing

```
float speedup_testing(r, S_A, S_B)
begin
    float γ = 1;
    boolean h = 1;
    while h do
        if HPT(S_A/γ, S_B) ≥ r then
            h = 1;
            γ = γ + 0.01;
        end
        else break;
    end
    return γ;
end
```

4.3 An Example of Quantitative Performance Comparison

In this section, the quantitative performance comparison of two commodity computers, X (Linux OS, 4-core 8-thread Intel i7 920 with 6 GB DDR2 RAM) and Y (Linux OS, 8-core AMD Opteron 8220 with 64 GB DDR2 RAM) is presented as an example of applying the HPT. In our experiments, each SPLASH-2 benchmark (eight threads) is repeatedly run five times on each computer, using default workloads of SPLASH-2. By specifying the speedup-under-test γ to be 1.76, we use the HPT to test how reliable the proposition "Computer X is more than 1.76 times faster than Computer Y" is over SPLASH-2. Testing such a proposition is equivalent to testing "Computer $\tilde{X}$ is faster than Computer Y" over SPLASH-2, where $\tilde{X}$ is a virtual computer whose performance scores are always 1/1.76 of the corresponding scores of the real computer X. Table 3 presents the details of the comparison. To be specific, all performance scores are normalized to the first run of computer Y on each benchmark. In conducting quantitative comparison, we divide all performance scores of X by 1.76 times (we store these reduced scores in $S_{\tilde{X}}$), and utilize the HPT to compare the reduced scores against those of computer Y (stored in S_Y). For the τth benchmark ($\tau = 1, \ldots, n$), "Stat. Win." indicates the winner whose performance on the τth benchmark is significantly (with a 0.95 confidence) better. We indicate "$\tilde{X}$" if the reduced performance of X still wins, and we indicate "Y" if the performance of Y wins over the reduced performance of X. In case there is no definite winner, we indicate "Tie". "Med." indicates the median of the five performance scores (of A and B), "Diff." shows the (significant) difference between the median performance scores of A and B, "Rank" shows the rank of the absolute value of d_τ. According to the HPT, the virtual computer $\tilde{X}$ beats computer Y significantly on eight benchmarks, ties on two benchmarks, loses on four benchmarks. Following the flow introduced in Section 4.1, the HPT concludes that "Computer $\tilde{X}$ is faster than Computer Y with a 0.95 confidence", suggesting that "Computer X is more than 1.76 times faster than Computer Y with a 0.95 confidence" (i.e., the 0.95-Speedup of Computer X over Computer Y is 1.76 over all SPLASH-2 benchmarks).

5 Experimental Evaluations

5.1 Comparisons Using the Geometric Mean Performance

In traditional quantitative comparisons, geometric mean of the performance scores of a computer over different benchmarks is often utilized to estimate the performance speedup of one computer over another. In most cases, such comparison results, presented without confidence estimates, are unreliable. We perform the following more extensive experiments: we collect SPEC CPU2006 reports of 14 computer systems (named as A1, A2, ..., G1, G2 for short) and SPEC MPI2007 reports of another 14 computer systems (named as H1, H2, ..., M1, M2 for short) from SPEC.org [1]. We analyze both the 0.95-Speedup (performance speedup estimated by the HPT, with the guaranteed confidence 0.95) and GM-Speedup for each computer pair, and present results in Table 4. It can be observed from Table 4 that the GM-Speedup is higher than the 0.95-Speedup on all 14 pairs of computer systems, and the largest error between the GM-Speedup and 0.95-Speedup can be 56.3 percent. Meanwhile, compared with the acceptable confidence 0.95, the loss of confidence brought by the unreliable GM-Speedup ranges from 12.6 to 89.5 percent, showing that all 14 GM-Speedups are rather unreliable.

TABLE 4
Quantitative Performance Comparisons Based on SPEC CPU2006 (Top) and SPEC MPI2007 (Bottom), Where 0.95-Speedup Is Obtained by the HPT, GM-Speedup Is Obtained by Comparing the Geometric Mean SPEC Ratios of Computers, and HPT-Confidence Is Estimated by the HPT

Part I: **SPEC CPU2006**	A1-A2	B1-B2	C1-C2	D1-D2	E1-E2	F1-F2	G1-G2
0.95-Speedup	2.64	2.24	1.39	2.45	1.76	1.54	1.15
HPT-Confidence	0.95	0.95	0.95	0.95	0.95	0.95	0.95
GM-Speedup	3.35	3.50	1.70	3.26	1.98	1.67	1.27
Speedup Error	*+26.9%*	*+56.3%*	*+22.3%*	*+33.1%*	*+12.5%*	*+8.4%*	*+10.4%*
HPT-Confidence	0.18	0.31	0.33	0.17	0.12	0.68	0.15
Confidence Loss	*-81.1%*	*-67.4%*	*-65.3%*	*-82.1%*	*-87.4%*	*-28.4%*	*-84.2%*
Part II: **SPEC MPI2007**	H1-H2	I1-I2	J1-J2	K1-K2	L1-L2	M1-M2	N1-N2
0.95-Speedup	1.63	1.87	1.92	1.34	2.15	1.12	1.70
HPT-Confidence	0.95	0.95	0.95	0.95	0.95	0.95	0.95
GM-Speedup	1.76	2.09	2.10	1.54	2.39	1.33	1.94
Speedup Error	*+8.0%*	*+11.4%*	*+9.7%*	*+15.1%*	*+11.3%*	*+19.0%*	*+14.3 %*
HPT-Confidence	0.39	0.10	0.42	0.58	0.63	0.83	0.53
Confidence Loss	*-58.9%*	*-89.5%*	*-55.8%*	*-38.9%*	*-33.7%*	*-12.6%*	*-44.2%*

5.2 Comparisons Using t-Test

Let us first recall the example presented in Section 3.3. As clearly shown by Table 2, BL265+ outperforms CELSIUS R550 on all benchmarks of SPECint2006. Moreover, performance distributions of both computers are non-normal due to the outlier performance on *libquantum*, and sizes of both performance samples are only 12 (far from enough for invoking the CLT), which suggest that paired t-test is not applicable here. However, if voluntarily ignoring that fact and sticking to paired t-test, we get the incorrect and counter-intuitive conclusion "BL265+ does not significantly outperform CELSIUS R550 at the confidence level 0.95".

In contrast, the HPT is much more robust in the presence of performance outliers. According to the HPT, BL265+ outperforms CELSIUS R550 with the confidence 1, which well complies with our intuition. In addition, the HPT also concludes that BL265+ is 1.41 times faster than CELSIUS R550 with the confidence 0.95, i.e., the 0.95-speedup of BL265+ over CELSIUS R550 is 1.41.

In the rest of this section, the outputs (confidence or r-speedup) of HPT are quantitatively compared against outputs of paired t-test. From a statistical viewpoint, we are conducting "apple-to-orange" comparisons because HPT and t-test are built on different statistics. Regardless of the statistical rigor, however, such comparisons are still meaningful since they may reveal how incorrect usage of t-test misleads the belief of practitioners.

Following the last section, our empirical study still analyzes SPEC CPU2006 scores of 14 computer systems (A1, A2, ..., G1, G2), and SPEC MPI2007 scores of another 14 computer systems (H1, H2, ..., N1, N2). By fixing the performance speedup for each pair of computers, Table 5 compares confidences obtained by the HPT with those obtained by paired t-test. For the first seven computer pairs, t-test rejects five out of the seven speedup results obtained by the HPT, due to the inaccurate t-Confidence. Interestingly, for the second seven computer pairs, HPT-Confidence and t-Confidence match much better, and t-test only rejects three out of the seven speedup results obtained by the HPT.

We can easily explain the above observation after checking the normality of the data. While SPEC CPU2006 scores of the first 14 computer systems (A1, A2, ... , G1, G2) are generally not distributed normally according to Lilliefors test [20], there is no significant statistical evidence that SPEC MPI2007 scores of any of the second 14 computer systems (H1, H2, ... , N1, N2) are *not* distributed normally. In other words, the usage of t-test is incorrect for the first seven computer pairs, but is correct for the second seven computer pairs. When t-test is correctly used, it is not strange that confidences provided by the HPT and t-test almost coincide with each other. A similar trend can also be observed from Table 6, where we compare performance speedups obtained by t-test against those obtained by HPT at the same confidence level 0.95. For the first seven computer pairs, the speedup error of paired t-test ranges from 1.4 to 21.6 percent, which highlights the impact of using inappropriate statistical techniques on the outcome of performance comparisons. For the second seven computer pairs, speedups provided by the HPT and t-test again coincide well with each other.

TABLE 5
Comparisons of Confidences Obtained by the HPT and Paired t-Test (Top: SPEC CPU2006; Bottom: SPEC MPI2007), Where HPT-Confidence Is Obtained by the HPT, and t-Confidence Is Obtained by the Paired t-Test

Part I: **SPEC CPU2006**	A1-A2	B1-B2	C1-C2	D1-D2	E1-E2	F1-F2	G1-G2
Speedup	2.64	2.24	1.39	2.45	1.76	1.54	1.15
HPT-Confidence	**0.95**	**0.95**	**0.95**	**0.95**	**0.95**	**0.95**	**0.95**
t-Confidence	0.91	**0.95**	**0.96**	0.94	0.89	0.87	0.52
Confidence Loss	*-4.2%*	*0%*	*0%*	*-1.1%*	*-6.3%*	*-8.4%*	*-45.3%*
Part II: **SPEC MPI2007**	H1-H2	I1-I2	J1-J2	K1-K2	L1-L2	M1-M2	N1-N2
Speedup	1.63	1.87	1.92	1.34	2.15	1.12	1.70
HPT-Confidence	**0.95**	**0.95**	**0.95**	**0.95**	**0.95**	**0.95**	**0.95**
t-Confidence	0.93	**0.96**	**0.96**	**0.96**	**0.96**	0.93	0.94
Confidence Loss	*-2.1%*	*0%*	*0%*	*0%*	*0%*	*-2.1%*	*-1.1%*

According to the statistical convention, the confidences in bold are the acceptable ones (≥ 0.95), and the rest are unacceptable ones (> 0.95).

TABLE 6
Comparisons of 0.95-Performance Speedups Obtained by the HPT and Paired t-Test (Top: SPEC CPU2006; Bottom: SPEC MPI2007), Where Each Speedup Has a Confidence of 0.95 for Each Technique

Part I: **SPEC CPU2006**	A1-A2	B1-B2	C1-C2	D1-D2	E1-E2	F1-F2	G1-G2
0.95-Speedup (HPT)	2.64	2.24	1.39	2.45	1.76	1.54	1.15
0.95-Speedup (Paired t-test)	2.07	2.28	1.41	2.39	1.59	1.38	1.04
Speedup Error	*-21.6%*	*+1.8%*	*+1.4%*	*-2.45%*	*-9.7%*	*-10.4%*	*-9.6%*
Part II: **SPEC MPI2007**	H1-H2	I1-I2	J1-J2	K1-K2	L1-L2	M1-M2	N1-N2
0.95-Speedup (HPT)	1.63	1.87	1.92	1.34	2.15	1.12	1.70
0.95-Speedup (Paired t-test)	1.60	1.87	1.93	1.35	2.16	1.09	1.67
Speedup Error	*-1.8%*	*0%*	*+0.5%*	*+0.7%*	*+0.5%*	*-2.7%*	*-1.8%*

The insights gained here are two-fold. First, it is important to choose appropriate statistical techniques to fit statistical characteristics of data. Second, the HPT is very robust against the validity of normality, and is a promising solution to performance comparisons in the presence/absence of normality.

6 Single-Number Performance Metric?

Using a single-number performance metric for each computer is the most intuitive way of conducting cross-application performance comparisons of computers, where the metric can be geometric mean performance, harmonic mean performance, arithmetic mean performance and so on. In such a comparison, the computer with a larger value of the single-number metric is considered to outperform the counterpart with a smaller value of the same metric, and the cross-application speedup of the first over the second is directly estimated as the quotient between their values under the metric.

Although single-number performance metrics have been widely used in computer architecture research, their effectiveness must be reconsidered when we deem the confidence to be a critical dimension of performance comparisons. In our viewpoint, each comparison result must be provided with the corresponding confidence, no matter which single-number performance metric is utilized by the performance comparison. However, in practice there are cases in which the usage of a single-number performance metric excludes rigorous estimate of confidence. More specifically, in common scenarios of computer architecture research where the number of performance observations is small (e.g., 10-20), rigorously estimating confidence of a comparison result induced by a single-number metric can even be plainly impossible. For example, when the performance distribution is not log-normally distributed and the number of benchmark applications is small (e.g., 10-20), it is infeasible to achieve a normally distributed sample mean of the logarithms of performance observations, thereby the confidence of the comparison using *geometric mean performance* cannot be rigorously estimated with t-statistics. Other statistical techniques like permutation test or bootstrap require even more performance observations, which are not applicable either. Under the above circumstance, the confidence of the comparison using geometric mean performance is hard (if possible) to be rigorously estimated, and the risk of drawing incorrect conclusion cannot be controlled. In order to gain statistical rigor, researchers may either give up the traditional metric "geometric mean", or increase the number of performance observations to make the CLT applicable (cf. Section 3.2).

In contrast, the HPT works well given small performance samples, and enables statistically rigorous cross-application comparison without explicitly using a single-number performance metric of each computer. It conducts a performance comparison between two computers using nonparametric statistics of the *performance gap* between two computers, and outputs a quantitative comparison result as 0.95-speedup of one computer over another. Embedded with the confidence information, the comparison result would be much more reliable than results offered by single-number metrics in many cases. Our central claim is that the HPT is a statistically rigorous replacement of traditional single-number performance metrics under various scenarios of performance comparisons.

7 Related Work

Performance comparisons of computers traditionally rely upon one single-number metric (e.g., geometric mean and harmonic mean) [22], [29], [30], [31], [32], though this approach can be rather unreliable. Having realized the importance of statistical inference, Lilja suggested to introduce several parametric statistical methods (e.g, confidence interval) to evaluate computer performance [12]. Alameldeen and Wood carried out in-depth investigations on the performance variability of multi-threaded programs, and they suggested to use the confidence interval and t-test, two parametric techniques, in order to address the issue of variability [4]. Sitthi-amorn et al. found that computer performance is mostly normally distributed on SPEC CPU2000 [13]. Similar observations were made in Java performance evaluation conducted by Georges et al. [21], who found that uni-application performance (SPECjvm98) on several single-core computers can, in general, be characterized using normal distributions. While valid, their observation does not seem to generalize to the broader case of multi-core systems and multi-threaded applications, based on our own experiments. Iqbal and John's empirical study [33] generally supported the log-normality for characterizing the SPEC performance of computers. However, their experiments were conducted after removing all "outlier benchmarks" in SPEC CPU2006. They also proposed a performance ranking system [33]. But unlike Wilcoxon test which uses rank information to construct statistics for computing confidence, the system directly offers a performance ranking without presenting the corresponding confidence.

In computer architecture research, statistical techniques have already been used to cope with various issues other than performance comparisons. For instance, statistical techniques were used for sampling simulation [34], principal components analysis was used to evaluate the representativeness of benchmarks [3], [35], and regression techniques were used to model the design space of processors [36], [37], [38], [39], [40]. Therefore, the computer architecture community is already largely familiar with complex statistical tools, so that embracing a more rigorous performance observation and comparison process is only a logical extension of the current trend.

8 Conclusion

In this paper, we formulate performance comparisons of computers as statistical tasks, and highlight the importance and impact of variability in performance observations and comparisons, as well as the risk of inappropriately using traditional single-number performance metric and t-statistics. We empirically check conditions under which t-statistics work appropriately, and reveal that t-statistics do not suit current practices of computer performance comparisons.

We propose the HPT framework for achieving both a rigorous and practical comparison of computer performance. In HPT, we adopt non-parametric SHTs that require neither normal performance distributions nor sufficiently large numbers of performance observations. Besides the benefits for performance comparisons, we have implemented the HPT as an easy-to-use open-source software and integrated the framework in PARSEC 3.0 benchmark suite [11], requiring no mathematical background.

Acknowledgments

The authors are grateful to Babak Falsafi, Shuai Hu, and to the anonymous reviewers for their helpful comments and insightful suggestions. This work was partially supported by the National Natural Science Foundation of China (under Grants 61100163, 61133004, 61222204, 61221062, 61303158), the 863 Program of China (under Grant 2012AA012202), the Strategic Priority Research Program of the CAS (under Grant XDA06010403), and the 10,000 talent program. Yunji Chen is the corresponding author of this paper. A preliminary version of this work appeared in the Proceedings of 18th International Symposium on High-Performance Computer Architecture (HPCA'12).

References

[1] (2013). [Online]. Available: http://www.spec.org/
[2] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The Splash-2 programs: Characterization and methodological considerations," in *Proc. 22nd Annu. Int. Symp. Comput. Arch.*, 1995, pp. 24–36.
[3] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: Characterization and architectural implications," in *Proc. 17th Int. Conf. Parallel Archit. cCompilation Techn.*, 2008, pp. 72–81.
[4] A. R. Alameldeen and D. A. Wood, "Variability in architectural simulations of multi-threaded workloads," in *Proc. 9th Int. Symp. High-Perform. Comput. Archit.*, 2003, pp. 7–18.
[5] Y. Chen, W. Hu, T. Chen, and R. Wu, "LReplay: A pending period based deterministic replay scheme," in *Proc. 37th Annu. Int. Symp. Comput. Arch.*, 2010, pp. 187–197.
[6] P. Montesinos, M. Hicks, S. T. King, and J. Torrellas, "Capo: A software-hardware interface for practical deterministic multiprocessor replay," in *Proc. 14th Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, 2009, pp. 73–84.
[7] T. Mytkowicz, A. Diwan, M. Hauswirth, and P. F. Sweeney, "Producing wrong data without doing anything obviously wrong!" in *Proc. 14th Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, 2009, pp. 265–276.
[8] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *J. Mach. Learn. Res.*, vol. 7, pp. 1–30, 2006.
[9] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics Bull.*, vol. 1, no. 6, pp. 80–83, 1945.
[10] (2013). [Online]: Available: http://novel.ict.ac.cn/tchen/hpt
[11] (2013). [Online]: Available: http://parsec.cs.princeton.edu/parsec3-doc.htm
[12] D. J. Lilja, *Measuring Computer Performance: A Practitioners's Guide*. Cambridge, U.K.: Cambridge Univ. Press, 2000.
[13] P. Sitthi-amorn, D. A. B. Weikle, and K. Skadron, "Exploring the impact of normality and significance tests in architecture experiments," in *Proc. Workshop Model., Benchmarking Simul.*, 2006.
[14] J. Suh and M. Dubois, "Dynamic MIPS rate stabilization in out-of-order processors," in *Proc. 36th Annu. Int. Symp. Comput. Archit.*, 2009, pp. 46–56.
[15] T. Bergan, O. Anderson, J. Devietti, L. Ceze, and D. Grossman, "Coredet: A compiler and runtime system for deterministic multithreaded execution," in *Proc. 15th Archit. Support Program. Lang. Oper. Syst.*, 2010, pp. 53–64.
[16] Y. Chen, T. Chen, L. Li, L. Li, L. Yang, M. Su, and W. Hu, "LDet: Determinizing asynchronous transfer for post-silicon debugging," *IEEE Trans. Comput.*, vol. 62, no. 9, pp. 1732–1744, Sep. 2013.
[17] K. K. Pusukuri, R. Gupta, and L. N. Bhuyan, "Thread tranquilizer: Dynamically reducing performance variation," *ACM Trans. Archit. Code Optim.*, vol. 8, no. 4, Article 46, 2012.
[18] C. Curtsinger and E. D. Berger, "Stabilizer: Statistically sound performance evaluation," in *Proc. 18th Archit. Support Program. Lang. Oper. Syst.*, 2013, pp. 219–228.
[19] E. Parzen, "On estimation of a probability density function and mode," *Ann. Math. Stat.*, vol. 33, no. 3, pp. 1065–1076, 1962.
[20] H. W. Lilliefors, "On the Kolmogorov-Smirnov test for normality with mean and variance unknown," *J. Amer. Stat. Assoc.*, vol. 62, no. 318, pp. 399–402, 1967.
[21] A. Georges, D. Buytaert, and L. Eeckhout, "Statistically rigorous java performance evaluation," in *Proc. 22nd Annu. ACM SIGPLAN Conf. Object-Oriented Program. Syst. Appl.*, 2007, pp. 57–76.
[22] J. R. Mashey, "War of the benchmark means: Time for a truce," *ACM SIGARCH Comput. Archit. News*, vol. 32, no. 4, pp. 1–14, 2004.
[23] J. M. Chambers, W. S. Cleveland, B. Kleiner, and P. A. Tukey, *Graphical Methods for Data Analysis*. New York, NY, USA: Chapman and Hall, 1983.
[24] L. L. Cam, "The central limit theorem around 1935," *Stat. Sci.*, vol. 1, pp. 78–91, 1986.
[25] Y. Chen, Y. Huang, L. Eeckhout, G. Fursin, L. Peng, O. Temam, and C. Wu, "Evaluating iterative optimization across 1000 datasets," in *Proc. 2010 ACM SIGPLAN Conf. Program. Lang. Des. Implementation*, 2010, pp. 448–459.
[26] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *Proc. IEEE Int. Workshop Workload Charact.*, 2001, pp. 3–14.
[27] R. A. Johnson and G. Bhattacharyya, *Statistics: Principles and Methods*. Hoboken, NJ, USA: Wiley, 1986.
[28] R. V. Hogg and Elliot Tanis, *Probability and Statistical Inference*. Englewood Cliffs, NJ, USA: Prentice Hall, 2005.
[29] D. Citron, A. Hurani, and A. Gnadrey, "The harmonic or geometric mean: Does it really matter?" *ACM SIGARCH Comput. Archit. News*, vol. 34, no. 4, pp. 18–25, 2006.
[30] J. L. Hennessy and D. A. Patterson, *Computer Architecture, a Quantitative Approach*. San Mateo, CA, USA: Morgan Kaufmann, 2006.
[31] L. K. John, "More on finding a single number to indicate overall performance of a benchmark suite," *ACM SIGARCH Comput. Archit. News*, vol. 32, no. 1, pp. 3–8, 2004.
[32] J. E. Smith, "Characterizing computer performance with a single number," *Commun. ACM*, vol. 31, no. 10, pp. 1202–1206, 1988.
[33] M. F. Iqbal and L. K. John, "Confusion by all means," in *Proc. 6th Int. Workshop Unique Chips Syst.*, 2010.

[34] R. E. Wunderlich, T. F. Wenisch, B. Falsafi, and J. C. Hoe, "Smarts: Accelerating microarchitecture simulation via rigorous statistical sampling," in *Proc. 30th Annu. Int. Symp. Comput. Archit.*, 2003, pp. 84–97.
[35] A. Phansalkar, A. Joshi, L. Eeckhout, and L. K. John, "Measuring program similarity: Experiments with SPEC CPU benchmark suites," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, 2005, pp. 10–20.
[36] T. Chen, Q. Guo, K. Tang, O. Temam, Z. Xu, Z.-H. Zhou, and Y. Chen, "Archranker: A ranking approach to design space exploration," in *Proc. 41th Annu. Int. Symp. Comput. Archit.*, 2014.
[37] V. Desmet, S. Girbal, and O. Temam, "Archexplorer.org: A methodology for facilitating a fair comparison of research ideas," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, 2010, pp. 45–54.
[38] Q. Guo, T. Chen, Y. Chen, Z.-H. Zhou, W. Hu, and Z. Xu, "Effective and efficient microprocessor design space exploration using unlabeled design configurations," in *Proc. 22nd Int. Joint Conf. Artif. Intell.*, 2011, pp. 1671–1677.
[39] E. İpek, S. A. McKee, R. Caruana, B. R. de Supinski, and M. Schulz, "Efficiently exploring architectural design spaces via predictive modeling," in *Proc. 12th Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, 2006, pp. 195–206.
[40] B. C. Lee, J. Collins, H. Wang, and D. Brooks, "CPR: Composable performance regression for scalable multiprocessor models," in *Proc. 41st Annu. IEEE/ACM Int. Symp. Microarchit.*, 2008, pp. 270–281.

Tianshi Chen received the BS degree in mathematics from the Special Class for the Gifted Young, University of Science and Technology of China (USTC), Hefei, China, in 2005, and the PhD degree in computer science from the Department of Computer Science and Technology, USTC, in 2010. He is currently an Associate Professor with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China. His current research interests include computer architecture, parallel computing, and computational intelligence.

Qi Guo received the BS degree in computer science from the Department of Computer Science and Technology, Tongji University, Shanghai, China, in 2007, and the PhD degree in computer science from the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, in 2012. He is currently a Postdoctoral Research Associate with Department of ECE, Carnegie Mellon University. His current research interests include computer architecture, very large scale integration design, and verification.

Olivier Temam has obtained a PhD in Computer Science from University of Rennes in 1993. He has been Assistant Professor at University of Versailles from 1994 until 1999, and then Professor at University of Paris Sud until 2004. Since then, he has been a Senior Research Scientist at INRIA Saclay in Paris, where he heads the ByMoore group, and adjunct professor at Ecole Polytechnique. His research spans microarchitecture, simulation, compilation and programming models. He has co-authored over 100 publications in international conferences and journals.

Yue Wu received the BS degree in statistics from University of Science and Technology of China, Hefei, China, in 2010. He is currently working towards the PhD degree at the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China. His current research interests include computer architecture and computational intelligence.

Yungang Bao received the BS degree in computer science and technology from Nanjing University, Nanjing, China, in 2003, and the PhD degree in computer science from the Institute of Computing Technology (ICT), Chinese Academy of Sciences, Beijing, China, in 2008. He is currently an associate professor with ICT. His current research interests include computer architecture, operating system and system performance modeling and evaluation. He was the recipient of the 2013 CCF-Intel Young Faculty Researcher Program (YFRP) Award.

Zhiwei Xu received the PhD degree from the University of Southern California in 1987. He is currently a professor and CTO of the Institute of Computing Technology, Chinese Academy of Sciences. His editorial board services include the IEEE Transactions on Services Computing and Journal of Parallel and Distributed Computing. His research interests include grid computing, distributed operating systems, and high performance computer architecture. He is a senior member of the IEEE.

Yunji Chen graduated from the Special Class for the Gifted Young, University of Science and Technology of China, Hefei, China, in 2002. Then, he received the PhD degree in computer science from Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS), Beijing, China, in 2007. Currently, he is a professor at ICT. He was a chief architect of Godson-3 processor. His research interests include computer architecture and computational intelligence. He has authored or coauthored 1 book and over 50 papers in these areas.