An Experimental Study on Execution Time Variation in Computer Experiments

Conference Paper · March 2014

DOI: 10.1145/2554850.2555022

3 authors, including:

Paulo Eduardo Nogueira
Federal Institute of Education, Science and Technology of São Paulo
8 PUBLICATIONS **13** CITATIONS
SEE PROFILE

Rivalino Matias Jr.
Universidade Federal de Uberlândia (UFU)
144 PUBLICATIONS **1,206** CITATIONS
SEE PROFILE

An Experimental Study on Execution Time Variation in Computer Experiments

Paulo Eduardo Nogueira
Federal Institute for Education, Science, and Technology
Morrinhos, Brazil
paulo.nogueira@ifgoiano.edu.br

Rivalino Matias Jr.
School of Computer Science
Federal University of Uberlandia
Uberlandia, Brazil
rivalino@fc.ufu.br

Elder Vicente
Electrical Engineering Department
Federal University of Triangulo Mineiro
Uberaba, Brazil
elder@icte.uftm.edu.br

ABSTRACT
In computer experiments, many research works rely on the accuracy of measured programs' execution time. We observe that not all studies consider that repeated executions of the same program, under the same experimental conditions, may produce statistically significant different completion times. In this work, we experimentally demonstrate that several sources of OS Jitter affect the execution time of computer programs. We compare various execution time samples using three test protocols, which apply different statistical techniques. The results show that significant differences are detected in all evaluated scenarios.

Categories and Subject Descriptors
D.4.8 [**Operating Systems**]: Performance – *measurements*.

General Terms
Measurement, Performance, Experimentation

Keywords
Execution time variability, experiments, OS jitter

1. INTRODUCTION
In computer science, many research works depend on measuring programs' execution time [11]. For example, to evaluate the speedup of two algorithms implemented in a given system, the experimenter compares their respective execution times. Very important is the ability of reproducing the observed experimental results, in order to demonstrate their confidence levels.

We have no guarantee that repeated executions of the same computer program, under the same experimental conditions, will produce the same completion times. In fact, what we observe in practice, many times, is a significant variation in successive executions of the same program on the same machine. In experimental research, this variation is known as experimental error and is caused by uncontrollable factors [7]. In computer experiments, these factors may be either hardware or software related.

The stochastic nature of the execution time variations makes its magnitude hard to predict. As a consequence, various experimental works, especially those relying on execution times, fail in not handling this problem adequately. It is not uncommon to find out research works reporting results of computer experiments based on a single experiment run. Given the above-mentioned variation problem, the one-run approach is clearly not reliable, since the single observed execution time could deviate, significantly, from the most frequently values. The lack of rigor in dealing with experimental errors in computer experiments has been object of discussion in several previous works (e.g., [1], [4], [5], [6], [8], [9], and [11]).

In this work, we present an experimental study on computer program execution time variation. We consider a specific source of execution variation, which is called OS Jitter [12]. It is characterized by the influence that the operating system (OS) internal routines cause on the user-level applications' execution times. We experimentally demonstrate how different sources of OS Jitter affect the execution time of a typical high-performance computing (HPC) program. We also compare different samples of execution times, under different OS Jitter scenarios, with the test protocols described in [11]. We use these protocols to detect statistical significant differences in comparable execution times.

The rest of this paper is organized as follows. Section 2 presents related works. Section 3 shows the methods and materials used. Section 4 presents our experimental study. Section 5 discusses the obtained results. Finally, Section 6 presents our conclusions.

2. RELATED WORK
According to Touati *et al.* [11], both high-performance computing and computer simulation fields have difficulties in reproducing experimental results. The authors propose a statistical approach to compare the execution time of two versions of the same program with the same input data. Their proposal showed good results, being more accurate than previous approaches, given that it considers the necessary assumptions required to apply the selected statistical methods, which is commonly neglected in many works.

Georges *et al.* [1] review several methods to evaluate Java applications using metrics proposed in previous works. They found that preceding results are not supported by a more judicious statistical analysis. However, more recent works have employed statistic-based methodologies with better results. They analyzed Java applications in both transient and steady-state phases. The results show that there is a substantial variability in different samples of execution time, where 20% are observed for steady-state and 8% for transient state. They also evaluated methods, prevalent in the literature, and reported that their results could be misleading in up to 16% of the cases.

In [8], [4], and [5], the authors found that different operating system factors cause significant variability in execution times, thus affecting the reproducibility of results. Mytkowicz *et al.* [8] changed the linking order and the space size of UNIX environment variables, in order to evaluate their influences on the

Draft version.

The final version of this manuscript was published in the ACM Symposium on Applied Computing (ACM SAC 2014), March 2014.

programs' execution time. They found that changes in linking order affect significantly the magnitude of execution time variations. Also, they show that increasing the UNIX environment size degrades the programs' execution times. Mazouz *et al.* [4] evaluated the runtime variability in sequential and parallel applications. Their results demonstrate that parallel applications are significantly more susceptible to variable execution times. Mazouz *et al.* [5] evaluated the usage of thread affinity strategies, and found a notable variability in execution times when there is thread migration, which corroborates the findings discussed in Mazouz *et al.* [6].

Pusukuri *et al.* [9] show that execution times in NUMA multicore systems are highly sensitive to the operating system policies. They demonstrated that changing policies of memory and processor scheduling might affect the variability of execution times up to 98%.

As can be observed in the literature, there are many factors related to the operating system that affect experimental results in terms of execution time.

3. METHODOLOGY

3.1 Experimental Approach

We adopt the DOE (Design of Experiment) method [7] to plan and execute our experiments. This method requires controlled changes on selected factors of the system under study, in order to observe the effect of these changes on the response variable. In this work, the response variable is the execution time of a test program that performs a matrix multiplication algorithm commonly used in scientific computing applications (see Figure 1). The factors are different sources of OS Jitter that affect the programs' execution time.

```
01  ...
02  for (z = 0; z < 48; z++)
03    for (j = 0; j < 1000; j++)
04      for (i = 0; i < 1000; i++)
05        for (k = 0; k < 1000; k++)
06          if  (z == 0) then
07            R[j][i] = R[j][i] +  (M[j][k] * M[k][i]);
08          else
09            R[j][i] = R[j][i] + (T[j][k] * M[k][i]);
10    if  (z < (48 - 1)) then
11      for (j = 0; j < 1000; j++)
12        for (i = 0; i < 1000; i++)
13          T[j][i] = R[j][i];
14          R[j][i] = 0;
15        end for
16  end for
17  ...
```

Figure 1. Matrix multiplication algorithm.

Each controlled factor assumes different operating levels. A factor at a specified level is called a treatment [7]. For the setup of treatment combinations, we adopt the signal matrix method [2], which is arranged according to the Yates' order [7]. Solving the signal matrix, we have a ranking of individual and combined factors that are sorted by their influence degree on the program's execution time.

Our control group is composed of all treatments where sources of OS Jitter are present, like in typical environments of computer experiments. The experimental groups are the treatments that we control the levels of each OS Jitter sources (factors) evaluated. We replicate each treatment fifty times, in order to have a sample size large enough to ensure a proper estimation of experimental errors and to determine if the differences among treatments are statistically significant. The completion times are measured at the end of each treatment (test program) execution, and this time is approximately 10 minutes with no OS Jitter control.

To avoid that a treatment test influences the execution of the subsequent treatment, we restart the OS kernel right before starting the execution of a new treatment, making sure that each treatment test starts in a renewed OS environment. For each treatment, we disable the automatic CPU frequency regulation. This feature allows the operating system to change dynamically the processor frequency, affecting the execution time of programs.

3.2 Methods

To analyze the experimental results we apply two different approaches. In the first, we identify which samples of execution times are considered statistically different using the non-parametric Kruskal-Wallis test [3]. For this approach, we adopt a test protocol where we assume that execution times may be not independent and identically distributed. We compare the samples and consider them different if their execution times show a *p-value* less than 0.05 (α=5%).

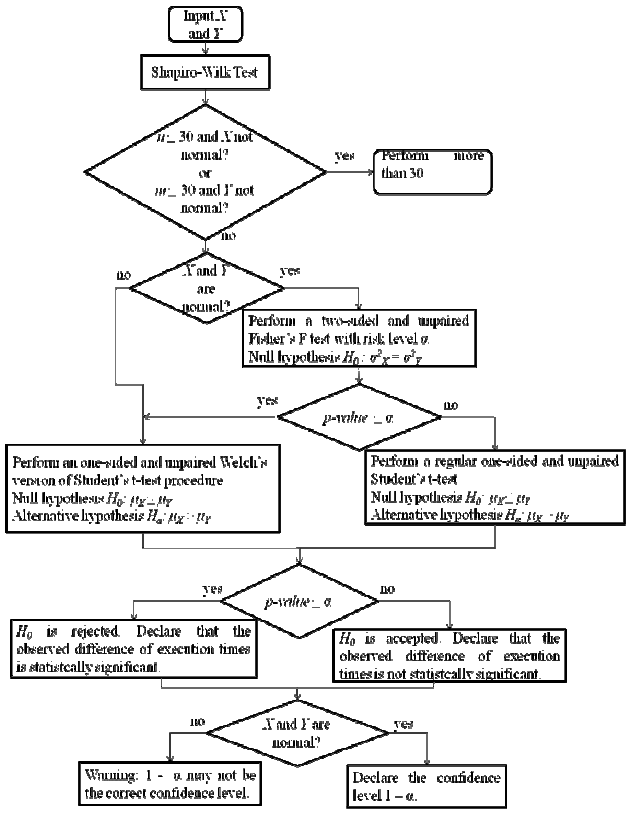

Figure 2. Protocol's steps for the average of execution times (adapted from [11]).

The second approach is based on two test protocols proposed in [11], which were developed to evaluate differences in program's execution times using the average and median values, respectively. Figure 2 shows the protocol applied to the average of execution times, which requires two samples of execution times (X and Y), where n and m are their respective sizes. For each paired comparison, X represents the samples with the highest

average value, and *Y* represents the samples with the lowest average. Based on a selected confidence level (1 – α), this protocol uses the Student's t-test to evaluate if the theoretical average of *X* is greater than *Y*'s average. Note that Student's t-test requires that both samples follow a Normal distribution and have the same variance. Hence, these assumptions are evaluated through the Shapiro-Wilk and Fisher's F tests, respectively. If the samples do not follow a Normal distribution or do not show the same variance, the protocol requires the use of the Welch's t-test.

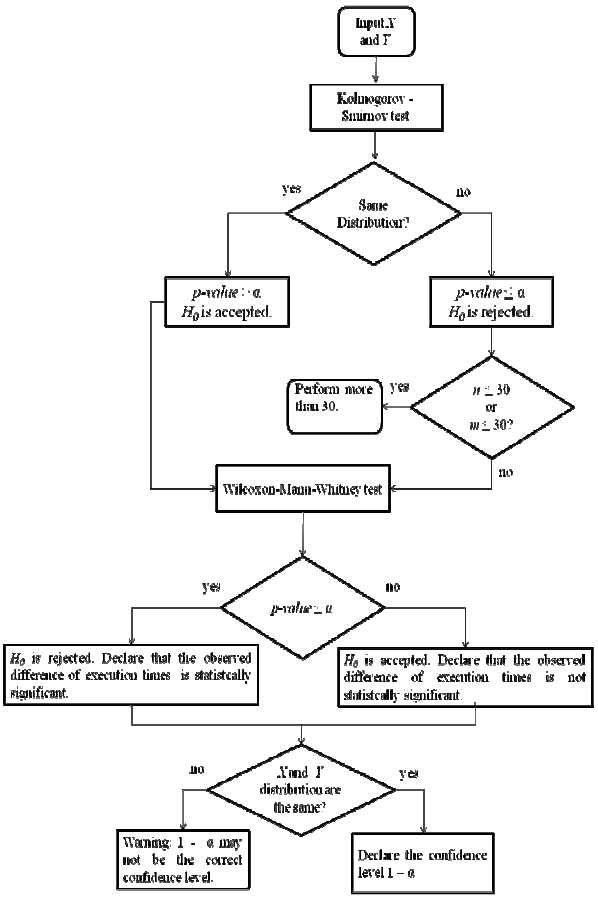

Figure 3. Protocol's steps for the median of execution times (adapted from [11]).

Analogous steps are followed in the protocol based on the median of execution times, whose steps are shown in Figure 3. In this case, *X* and *Y* are respectively the samples with the highest and lowest median values. We use the Wilcoxon-Mann-Whitney test with confidence level (1 – α) to detect if the difference of both samples is significant. The Wilcoxon-Mann-Whitney test requires that both samples (*X* and *Y*) follow the same distribution, so we test this assumption through the Kolmogorov-Smirnov test.

To keep both approaches consistent each other, in all three test protocols we adopt α equals to 5%.

4. EXPERIMENTAL STUDY

4.1 Test-bed & Instrumentation

Our test bed is based on a computer composed of two quad-core processors. Figure 4 shows the topology of the computer's processors with their three cache levels. The L3 cache is shared by all cores in the same processor, while each core has two individual caches (L1 and L2). For the sake of simplicity, we refer to each core as P0 to P7. The test program runs only on P1, where we rigorously control the enabling and disabling of OS Jitter sources. The remaining cores are used according to each treatment specification.

We use the Linux OS (version 2.6.32.28) in this study. Given that we control different sources of OS Jitter, it requires making changes in the Linux kernel.

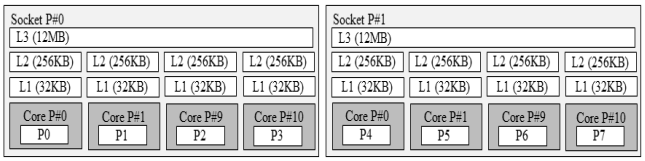

Figure 4. Topology of testbed's processors.

4.2 Experimental Planning

We encode each evaluated source of OS Jitter (factor) using upper case letters (e.g., see Table 1). Based on the DOE method used, we choose a full factorial design with two levels [7], where the levels are represented by symbols (+) and (-). The level (-) means that a given source of OS Jitter is disabled, and level (+) means the opposite. Hence, our factorial design with *K* factors, where each factor is evaluated at two levels, gives us a number of 2^K treatments. Each treatment's test is replicated *r* times, giving $2^K \times r$ test runs. We adopt *r* equals to fifty for all experiments.

4.3 Design of Experiment #1

This experiment evaluates five sources (factors) of OS Jitter, which are listed in Table 1.

Table 1. Factors and levels evaluated

			Level (-)	Level (+)
Factors	A	Runlevel	5	1
	B	Kernel Timers	Off	On
	C	IRQ Handling	Off	On
	D	Processor Affinity	Off	On
	E	Timer Interrupt Handling	Off	On

Factor A represents the OS runlevel [13]. It defines the services loaded during the OS initialization. At level (-) the runlevel is 5 (maximal number of service loaded), which is the default for many Linux OS environment; the level (+) sets the runlevel to 1 (minimal number of services). The OS Jitter effect of this factor on the execution time is related to the processes running concurrently with the user application, and consequently competing for resources (e.g., CPU and processor cache).

Factor B represents OS kernel timers, which are created in kernel space to execute routines in a scheduled way. Level (-) indicates kernel timers disabled on the processor (P1) that executes the test program. Level (+) allows kernel timers execute on P1.

Factor C represents different hardware interrupt requests (IRQ). Level (-) indicates that P1 cannot receive interrupt requests (except timer interrupts); in this case all IRQs are redirected to P0. At level (+) it allows IRQs to be handled on P1, which affects the test program execution.

Factor D relates to processor affinity implemented by the OS CPU scheduler. Level (-) means that processor affinity is disable, and

thus all processes can be executed in any processor; so they can be scheduled to run on P1. Level (+) means that all processes may run only on P0, not sharing P1 with the test program.

Factor E relates to the timer interrupt. At level (-) this interrupt is disabled for P1 and thus not interfering, periodically, on the execution time of the test program, as occur in level (+).

4.4 Design of Experiment #2

This second experiment adds a sixth factor to the setup used in experiment #1, so now we have 64 (2^6) treatments.

This additional factor is a background workload that shares L3 cache with the test program running on P1. The main purpose of this experiment is to make the OS memory management subsystem to deal with an increase in L3 cache misses, allowing us to measure this effect on the execution time of the test program. The letter F represents this new factor.

Factor F at level (-) means the background workload running on P5, which is not sharing L3 cache with P1. At level (+), we have the background workload running on P2, which shares L3 cache with P1. This allows us to represent scenarios where other processes share cache memory with the test program. We control the working set size of both competing processes to make sure they are large enough to fill the entire L3 memory cache (12 MB).

5. RESULTS

5.1 Experiment #1

In Figure 5, we observe that the variability of the execution times between the 1st and 16th treatments (G1) is quite lower than that obtained between the 17th and 32nd treatments (G2).

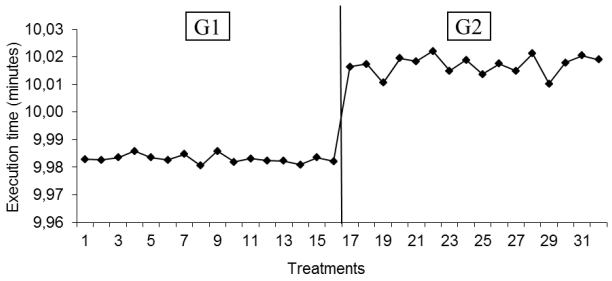

Figure 5. Average execution time for treatments of Exp. #1.

After the 16th treatment, there is a raise in the average execution time. Note that the factor E (timer interrupt) is disabled for all first sixteen treatments and enabled for the remaining treatments. This is an evidence of the influence of factor E on the execution time variability. Next, we calculate the percentage that each factor and their interactions contribute to the execution time variation of the test program. The results show that 91.23% of the variation is caused by factor E (timer interrupt) and the other factors did not show important contributions.

5.1.1 Execution time evaluation (1st approach)

Based on the 32 samples of execution time, obtained in Exp. #1, we perform the Kruskal-Wallis test and rejected H_0, which means that at least two samples are statistically significant different. Next, we conduct a comparison of samples and observe that every sample with factor E (timer interrupt) disabled is considered statistically different from samples with this factor enabled. This means there are significant differences between the groups of treatments (G1-G2) of Exp. #1.

Based on the results of paired comparisons of samples of treatments in the same group, we note that samples inside G1 present no significant differences when compared each other, and similar results are obtained for G2 with only 10% of comparisons considered statistically different. These results are presented in Table 2.

Table 2. Percentages of statistically significant paired comparisons of *X* and *Y*

	G1	G2
G1	0.0%	100.0%
G2		10.0%

5.1.2 Execution time evaluation (2nd approach)

Similar to the first approach, we select all samples from Exp. #1, where treatments with highest average execution time are considered as *X* and treatments with lowest average execution time are *Y*. Next, we conduct Shapiro-Wilk tests to evaluate if samples *X* and *Y* follow the Normal distribution.

Table 3 shows the percentages of paired comparison of samples analyzed through Student's and Welch's t-tests. We can see that Welch's t-test was prevalent in all comparisons. The adopted protocol defines that Student's t-test is only applied when both samples met the normality and homogeneity of variance assumptions. Otherwise, the Welch's t-test is used. This means that most of samples do not follow the Normal distribution.

Table 3. Percentages of paired comparisons per type of test

	Student's t-test	Welch's t-test
G1-G1	36.3%	63.7%
G1-G2	27.7%	72.3%
G2-G2	19.1%	80.9%

In order to verify which samples are statistically significant different, we perform all paired comparisons adopting the protocol procedures described in Figure 2. Table 4 shows the results. We detect significant differences in the execution times of the same program inside and outside the experimental groups.

Table 4. Percentages of comparisons statistically different (average based)

	G1	G2
G1	38.3%	100.0%
G2		77.5%

In addition to the average of execution times, we also tested the median values. Similar to the above-mentioned procedures for the average, using median we select all treatments from Exp. #1 with highest median as *X* and lowest median as *Y*.

Table 5. Percentages of paired comparisons with same distribution

	Same Distribution	Warning
G1-G1	100.0%	0.0%
G1-G2	99.6%	0.4%
G2-G2	97.7%	2.3%

According to procedures described in Figure 3, we apply the Kolmogorov-Smirnov test to both samples (*X* and *Y*). Table 5 shows the percentages of paired comparisons of execution times that follow or not the same distribution.

Next, we apply the Wilcoxon-Mann-Whitney test to all samples. We note that this test may not be accurate if the samples do not follow the same distribution [10], which may require a different alpha (α) value. As can be seen in Table 5, the tested samples predominantly follow the same distribution.

Table 6. Percentages of comparisons statistically different (median based)

	G1	G2
G1	50.8%	100.0%
G2		74.2%

Table 6 shows the results for comparisons of execution times in Exp. #1, considering the median values. We observe that there are differences in the results obtained with the average (see Table 4) and median (see Table 6), which are 12.5% for G1-G1 and 3.3% for G2-G2. Although the small differences, most of them comes from G1.

Given that the results based on the average are obtained mostly using non-Normal distribution, which means they may not be accurate, it seems using the median values is more reliable.

5.2 Experiment #2

The analysis of this experiment follows the same procedures described in Section 5.1. Figure 6 presents the average execution time for each treatment. We split the treatments in four groups (G1 to G4).

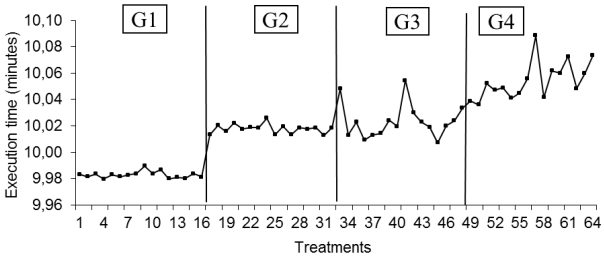

Figure 6. Average execution time for treatments of Exp. #2.

We observe that the variability of execution times increases according to the group. G1 and G2 reproduce the same treatments evaluated in Exp. #1. Factor F (shared L3 cache) is enabled in all treatments of G3 and G4, where in G3 the factor E (timer interrupt) is disabled and in G4 it is enabled. We can see that the individual contribution of factors E and F on the test program's execution time are similar. Factor F contributes with 33.90% and the factor E with 24.76% of the execution time variation. Note that the other factors did not show important contributions.

5.2.1 Execution time evaluation (1st approach)

Based on the Kruskal-Wallis test and the comparison of pairs of samples (see Table 7), this test shows no significant differences in the execution times, except in pairs 33-34, 33-36, and 33-37, from G3. Comparing the treatments in G2 to treatments in G3, we obtain only 4.3% of the comparisons considered statistically different. The comparisons among treatments from G1-G2, G1-G3, G1-G4, G2-G4, and G3-G4 show statistically significant differences.

Table 7. Percentages of statistically significant paired comparisons of *X* and *Y*

	G1	G2	G3	G4
G1	0.0%	98.0%	97.3%	100.0%
G2		0.0%	4.3%	75.4%
G3			7.5%	66.8%
G4				0.0%

5.2.2 Execution time evaluation (2nd approach)

Table 8 shows the percentages of comparison of samples of execution times analyzed through Student's and Welch's t-tests. We can see that Welch's t-test was again prevalent in all comparisons; it is used in 100% of the comparisons G1-G3, G1-G4, G2-G3, and G2-G4. Except to G2-G2 (81.6%), the Welch's t-test is applied to more than 90% of the analyses.

Table 8. Percentages of paired comparisons per type of test

	Student's t-test	Welch's t-test
G1-G1	1.6%	98.4%
G1-G2	5.1%	94.9%
G1-G3	0.0%	100.0%
G1-G4	0.0%	100.0%
G2-G2	18.4%	81.6%
G2-G3	0.0%	100.0%
G2-G4	0.0%	100.0%
G3-G3	0.8%	99.2%
G3-G4	0.8%	99.2%
G4-G4	1.2%	98.8%

Table 9 shows the results of all comparisons of inter and intra groups for the average of execution times. Table 10 shows the percentages of paired comparison of samples of execution times that follow or not the same distribution.

Table 9. Percentages of comparisons statistically different (average based)

	G1	G2	G3	G4
G1	66.6%	100.0%	100.0%	100.0%
G2		69.1%	59.7%	100.0%
G3			61.6%	90.6%
G4				75.8%

Table 11 shows the results for all comparisons of Exp. #2, for the median analysis. The results of this experiment are consistent with the observed in Exp. #1 in terms of differences between average and median execution times, which can be seen in Tables 9 and 11.

Table 10. Percentages of paired comparisons with same distribution

	Same Distribution	Warning
G1-G1	93.7%	6.3%
G1-G2	97.7%	2.3%
G1-G3	2.7%	97.3%
G1-G4	5.1%	94.9%
G2-G2	100.0%	0.0%

G2-G3	1.6%	98.4%
G2-G4	5.1%	94.9%
G3-G3	88.3%	11.7%
G3-G4	80.9%	19.1%
G4-G4	71.9%	28.1%

Table 11. Percentages of comparisons statistically different (median based)

	G1	G2	G3	G4
G1	68.3%	100.0%	100.0%	100.0%
G2		70.8%	58.2%	100.0%
G3			65.8%	92.2%
G4				69.2%

We highlight that the results based on the average (see Table 9) are obtained mostly from non-Normal distributions, thus they may not be so accurate according to the adopted protocol. Similar observation we make to some cases (G1-G3, G1-G4, and G2-G3) using median values (see Table 11), since their samples do not follow the same distribution.

6. CONCLUSION

In this work, we presented an experimental study on the variation of execution times in repeated computer program runs. We considered a specific cause of execution time variation known as OS Jitter phenomenon [12]. We experimentally tested how different sources of OS Jitter affect the execution time of a typical scientific computing program. We detected statistically significant variations, under various OS Jitter scenarios, using different test protocols. Note that the obtained experimental results corroborates the findings in [11], demonstrating that using the median values is more reliable than average values, especially in presence of large data variance or outliers.

All tests demonstrated that the ability to reproduce execution times in computer experiments is importantly affected by OS Jitter effects; computer experiments that rely on observed execution times should consider this influence in their experimental plans. Thus, we conclude that such experiments, when based on a single execution time or even a few ones, may present high chance of misleading results.

Based on the findings of this work and support from the literature [11], we recommend no less than 30 replications for each treatment test, where the median of the sample should be used instead of the average value.

Also, the experimental planning, execution, and result analysis procedures must be conducted following comprehensive protocols covering all above-mentioned details; in this work we demonstrated how to integrate three execution time test protocols with the DOE method, allowing to implement controlled experiments statistically planned.

7. REFERENCES

[1] Georges, A., Buytaert, D., and Eeckhout, L. Statistically rigorous java performance evaluation. In *Proceedings of the 22nd annual ACM SIGPLAN conference on Object-oriented programming systems and applications*, *(OOPSLA '07)*. ACM, New York, NY, USA, 2007, 57–76.

[2] Jain, R. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Wiley-Interscience, New York, NY, 1991.

[3] Kvam, P.H. and Vidakovic, B. *Nonparametric Statistics With Applications to Science and Engineering*. Wiley-Interscience, New York, NY, 2007.

[4] Mazouz, A., Touati, S.-A.-A., and Barthou, D. Study of variations of native program execution times on multi-core architectures. In *Proceedings of the 2010 International Conference on Complex, Intelligent and Software Intensive Systems*, *(CISIS '10)*. IEEE Computer Society, Washington, DC, USA, 2010, 919–924.

[5] Mazouz, A., Touati, S.-A.-A., and Barthou, D. Analysing the variability of openmp programs performances on multicore architectures. In *Proceedings of the 4th Workshop on Programmability Issues for Heterogeneous Multicores (MULTIPROG '11)*. Heraklion, Greece, 2011, 14.

[6] Mazouz, A., Touati, S.-A.-A., and Barthou, D. Performance evaluation and analysis of thread pinning strategies on multi-core platforms: Case study of SPEC OMP applications on intel architectures. In *Proceedings of the 2011 International Conference on High Performance Computing and Simulation (HPCS)*. IEEE, Istanbul, Turkey, 2011, 273-279.

[7] Montgomery, D. C. *Design and Analysis of Experiments*. John Wiley, 3rd edition, 2000.

[8] Mytkowicz, T., Diwan, A., Hauswirth, M., and Sweeney, P. F. Producing wrong data without doing anything obviously wrong! In *Proceedings of the 14th international conference on Architectural support for programming languages and operating systems*, *(ASPLOS XIV)*. ACM, New York, NY, USA, 2009, 265–276.

[9] Pusukuri, K. K., Gupta, R., and Bhuyan, L. N. Thread tranquilizer: Dynamically reducing performance variation. *ACM Transaction Architecture Code Optimization (TACO)*, 8, 4 (January 2012), 46:1-21.

[10] Sheskin, D. J. *Handbook of Parametric and Nonparametric Statiscal Procedures*.CRC, 3rd edition, 2003.

[11] Touati, S.-A.-A., Worms, J., and Briais, S. The speedup-test:a statistical methodology for programme speedup analysis and computation. *Concurrency and Computation: Practice and Experience*, 25, 10 (July, 2013), 1410–1426.

[12] Tsafrir, D., Etsion, Y., Feitelson, D. G. and Kirkpatrick, S. System noise, os clock ticks, and fine-grained parallel applications. In *Proceedings of the 19th annual international conference on Supercomputing, (ICS '05)*. New York, NY, 2005, 303–312.

[13] Van Vugt, S. *The Definitive Guide to Suse Linux Enterprise Server*. Apress, 2006.