Proceedings of the 2015 Winter Simulation Conference
L. Yilmaz, W. K. V. Chan, I. Moon, T. M. K. Roeder, C. Macal, and M. D. Rossetti, eds.

A QUANTITATIVE STUDY ON EXECUTION TIME VARIABILITY IN COMPUTING EXPERIMENTS

Paulo Eduardo Nogueira

Goiano Federal Institute
Morrinhos, GO, BRAZIL

Rivalino Matias Jr.

School of Computer Science
Federal University of Uberlandia
Uberlandia, MG, BRAZIL

ABSTRACT

Several modeling, simulation and experimental research works in computer science and engineering depend on correctly measuring the execution time of computer programs. It is observed that not everyone takes into account that repeated executions of the program with the same input can result in execution times statistically significant different. The lack of rigor in the analysis of execution times of computer programs has been investigated in several studies in the literature. In this work, we first reproduce experiments from the literature in order to analyze the statistical properties of their results in terms of execution times, as well as to assess the effects of different variability sources in influencing the execution times. Particularly, we consider variability sources related to the operating system. We also propose a protocol to systematize the comparison of programs' execution times in order to identify the significant differences in samples obtained from experiments with multiple treatments.

1 INTRODUCTION

In computer science and engineering, several research works depend on the analysis of execution times of computer programs (Touati, Worms, and Briais 2013). For instance, by experimentally evaluating the performance of two algorithms in a given computer system, the researcher compares the execution times of their implementations. In this case, it is crucial to consistently reproduce the experiments so as to obtain statistical confidence on the results.

In computer systems, not always do successive executions of the same program with the same input produce the same execution times. Actually, what is often seen in practice is a significant variation in execution times of the same program in successive executions. In experimental research, this variation is known as experimental error and is caused by uncontrollable factors (Montgomery 2000). In computing experiments, these factors may be related to hardware or software, such as the influence of hardware interrupt mechanisms, the cache memory architecture, or the different interferences caused by operating system (OS) routines known as OS jitter (Vicente and Matias Jr. 2013), among other factors. In this paper, we study the software factors related to OS.

The stochastic nature of the variations caused by the factors mentioned above makes their magnitude hard to predict. Consequently, several experimental works, particularly those that depend on the correct analysis of program execution times, fail when dealing with this issue. It is not uncommon to find research works that report results of computing experiments based on a single run of the experiment. Due to the variation problem above-mentioned, it is evident that analyzing the results of a single run is not a reliable approach, since the single execution time observed may significantly deviate from the most frequent values. The lack of rigor when dealing with experimental errors in computing experiments has been investigated in several studies in the literature (e.g., Georges, Buytaert, and Eeckhout 2007;

Mytkowicz et al. 2009; Mazouz, Touati, and Barthou 2010; Mazouz, Touati, and Barthou 2011a; Mazouz, Touati, and Barthou 2011b; Pusukuri, Gupta, and Bhuyan 2012; Touati, Worms, and Briais 2013).

In this paper, we present an experimental study on the variation of execution times of computer programs. We consider different sources of OS-related variation in execution times, namely: *runlevel* (Vicente and Matias Jr. 2013), *size of environment variables* (Mytkowicz et al. 2009), and *thread affinity* strategies (Mazouz, Touati, and Barthou 2011a). Complementarily, we improve the method of execution time analysis proposed in (Touati, Worms, and Briais 2013), offering better precision in the analysis of data from multiple experimental settings (treatments). The remaining of this paper is structured as follows. Section 2 presents the related works. Section 3 describes the method and materials used, as well as the experimental study planning. Section 4 discusses the experimental results, and Section 5 presents our conclusion and final remarks.

2 RELATED WORKS

According to Touati, Worms, and Briais (2013), different research areas in computer science have difficulties in reproducing experimental results when it comes to program execution times. The authors proposed a statistical approach to compare the execution times of two version of the same program with the same input. The proposal yields positive results, being more precise than previous works in the literature (e.g., Lilja 2005; Georges, Buytaert, and Eeckhout 2007).

Georges, Buytaert, and Eeckhout (2007) reviewed several methods to assess execution times of Java applications using metrics proposed in different studies. By using a more rigorous statistical analysis, the results of the previous studies were not confirmed. The authors also assessed prevailing methods in the literature and verified that their results could be wrong in up to 16% of the cases.

In (Mytkowicz et al. 2009; Mazouz, Touati, and Barthou 2010; Mazouz, Touati, and Barthou 2011a; Pusukuri, Gupta, and Bhuyan 2012), the authors observed that different factors of the operating system cause significant variability in program execution times, thus impacting result reproducibility. Mytkowicz et al. (2009) changed the linking order of object files and also the size of UNIX environment variables in order to assess their influences on program execution times. They found that the first change significantly impacted the magnitude of the variations in execution times. They also observed that increasing the size of OS environment variables degraded the execution times of the programs analyzed. Mazouz, Touati, and Barthou (2010) assessed the variability in execution times of sequential and parallel programs. The results showed that parallel programs were significantly more susceptible to variable execution times. Mazouz, Touati, and Barthou (2011a) assessed the use of different thread affinity strategies and found a notable variability in execution times when threads migrate across the processors, corroborating the results presented in Mazouz, Touati, and Barthou (2011b). Pusukuri, Gupta, and Bhuyan (2012) showed that execution times in multi-core systems are highly sensitive to the OS's processor management policy. They showed that changing such OS policy can reduce the variability of execution times in up to 98%.

As it can be seen in the literature, there are several OS-related factors that impact the experimental result in terms of execution time. Unlike from previous works, in this study we use a rigorous experimental method to confirm these influences. We analyze the variability of execution times through a protocol that considers experiments with multiple treatments and deals with the *familywise error rate* (FWER) problem, different from Touati, Worms, and Briais (2013) that did not cover these two important issues.

3 EXPERIMENTAL PLANNING

3.1 Method

In this study, we adopted the DOE (Design of Experiment) statistical method (Montgomery 2000) to plan and carry out our experiments as well as to analyze their results. This method requires controlled changes on the factors under study, so that the effects of these changes on the response variable can be accurately

measured. The response variable of interest in this study was the execution time of programs chosen for each test scenario (treatment). Each factor (source of execution time variability) was evaluated under different operation levels. A type of treatment is defined as a given combination of factors and levels (Montgomery 2000). In order to define the treatments to be evaluated, the signal matrix method (Jain 1991) configured according to Yates's order (Montgomery 2000) was used. To keep the execution of a treatment from influencing the results of the subsequent treatment, we restarted the OS for each new treatment execution.

Denote by X_{it} the sample of execution times of program i executed for the experimental condition defined in the treatment t, where X_{it} is a random variable with mean $\bar{x}_{it}$ and median $\tilde{x}_{it}$. Given that n is the sample size, then $X_{it} = \{x_{1it}, ..., x_{nit}\}$. Hence, for each pair of different treatments of a given experiment, e.g., t_1 and t_2, we compare their means and medians through hypothesis tests, where the null hypothesis, H_0, is respectively $\bar{x}_{it_1} = \bar{x}_{it_2}$ and $\tilde{x}_{it_1} = \tilde{x}_{it_2}$, and the alternative hypothesis, H_1, is respectively $\bar{x}_{it_1} \neq \bar{x}_{it_2}$ and $\tilde{x}_{it_1} \neq \tilde{x}_{it_2}$. This procedure is performed for all pairs of treatments for each experiment.

If a given hypothesis test shows a significantly statistical difference, then this indicates that one or more factors being controlled in the corresponding treatments exert an important influence on the execution times of the program under test. This shows that the experimental conditions of the evaluated treatments must be carefully considered when analyzing the execution times of computer programs under these same conditions, otherwise their results might be misinterpreted. We used the widely adopted program *time* (Kerrisk 2010) to measure the execution times of the programs under test. All experiments were executed in the Linux OS.

3.2 Design of Experiment #1

This experiment aimed to assess the effect of the factors *runlevel* and *compiler optimization* on the execution time of the NPB's benchmarks (NASA Advanced Supercomputing Divison 2014). The NPB is a set of benchmark programs that mimic the computation and data movement in typical CFD (computational fluid dynamics) applications. We used the NPB version 3.3.1, which is composed of 10 benchmark applications. More details about each NPB application can be obtained in (NASA Advanced Supercomputing Divison 2014). Table 1 summarizes the factors and levels adopted in this experiment.

Table 1: Factors and levels evaluated in Exp #1.

		Level (-)	**Level (+)**
Factors	Runlevel (RL)	5	3
	Optimization (O)	O2	O3

The *runlevel* defines a given configuration setup for the OS. In our experiments, at level (-), the *runlevel* factor assumed the value 5, which indicates a larger number of OS administrative processes (services) running in background. On the opposite, the level (+) sets this factor to 3, which results in a lower number of administrative processes. By varying this factor, we want to verify the effect of the OS administrative processes on the NPB's execution time. The optimization factor refers to the compiler optimization applied to each NPB benchmark. In this study, we used the *gcc* compiler version 4.7.2 (Stallman and The GCC Developer Comunity 2012). At level (-), the compiler was set to optimization O2, while at level (+) it used optimization O3, which has a more aggressive optimization than O2 and results in an larger binary code due to the extensive use of inline functions, among other optimizations (Stallman and The GCC Developer Comunity 2012).

In this experiment, each treatment was replicated 31 times, and the first replication was discarded so that the analysis was carried out with reduced or none influence of disk buffer/cache. The replications

aimed to provide a large enough sample that yields an appropriate estimate of experimental errors, thus helping to determine whether the differences among the treatments are statistically significant or not.

3.3 Design of Experiment #2

This experiment reproduced the experiment carried out in (Mytkowicz et al. 2009), whose goal was to assess the effect of OS *environment variable size* (EVS) on the execution time of NPB's benchmarks. In general, this is a non-considered factor when experimenters plan and analyze computing experiments. In (Mytkowicz et al. 2009), the authors reported that changes in the size of OS *environment variables* impact the program stack alignment and, consequently, the alignment of structures allocated in the process' heap, which influences the execution time of programs.

In our study, each NPB benchmarks ran 31 times, for each OS *environment variable size* evaluated. For the same reason as in the previous experiment, the first run of each EVS treatment was discarded. The OS *environment variable* used was created for this purpose and received a string value whose size varied as follows: 0, 64, 128, 256, 512, 1024, 2048, and 4096 bytes. The Linux feature of initial memory stack address randomization (Shacham et al. 2004) was disabled in order to individually assess only the effects of different EVS. All NPB benchmarks were compiled with optimizations O2 and O3, as in Exp. #1. In terms of OS *runlevel*, we adopted the value 1 so as to have the lowest possible number of administrative processes running in background during the treatments, aiming at reducing their influence on the results.

3.4 Design of Experiment #3

Experiment #3 was performed to assess the *thread affinity* factor, which represents the OS allocation strategy of multiple threads across the machine processors (or cores). This experiment reproduced the one carried out in (Mazouz, Touati, and Barthou 2011a), aiming to analyze the influence of *thread affinity* on the variation of the programs' execution times. In (Mazouz, Touati, and Barthou 2011a), the compiler *icc* (INTEL 2014) was used, which implements the following strategies: *no affinity*, *compact*, and *scatter*. In the first strategy, the OS is free to allocate threads among the processors according to their availability. In the second strategy, the OS is instructed to allocate the threads in each core sequentially, as close as possible, in order to increase the chances of sharing the processors' cache. In the third, the OS is instructed to distribute the threads among the processors as uniformly as possible.

In this study, we could reproduce these three *thread affinity* strategies by using the features implemented by *libgomp* (OpenMP) (GNU 2014) and *gcc*. In this case, the *thread affinity* strategy was chosen using the environment variable GOMP_CPU_AFFINITY. The setting of *gcc+libgomp* compatible with *icc-compact* is "GOMP_CPU_ AFFINITY=0-7". To implement *icc-scatter*, we used "GOMP_ CPU_AFFINITY=0 4 2 6 1 5 3 7". We compiled the OpenMP version for each NPB benchmark and ran each one 31 times at *runlevel* 1, and the first execution was discarded as in the other experiments. Table 2 shows the levels and factors for this experiment. Since each factor was assessed at three levels, 3^2 treatments were carried out.

Table 2: Factors and levels evaluated in Exp. #2.

		Levels
Factors	Number of Threads (NT)	2, 4, and 6
	Strategies of Affinity (SA)	*No Affinity*, *Compact*, and *Scatter*

3.5 Protocol for Analysis of Execution Times

Examining the approach described in (Touati, Worms, and Briais 2013), which assesses the statistical significance of the variation of two execution time samples, we verified that the proposed method is inappropriate to deal with experiments in which the samples of execution times are obtained from multiple treatments, i.e., to compare samples from different experimental settings. This occurs because the tests used in (Touati, Worms, and Briais 2013) to assess the statistical significance of the differences in execution times, using mean and median, the *Student's t-test* and *Wilcoxon-Mann-Whitney test* (Sheskin 2003), respectively, are appropriate to compare only two samples.

Employing these tests to compare samples from multiple treatments may lead to the problem known as *familywise error rate* (Howell 2010), which increases the likelihood of *Type I error* (Sheskin 2003), i.e., the probability of rejecting the null hypothesis (H_0) when it is true. In the two above-mentioned statistical tests, H_0 indicates that there is no statistically significant difference in execution times; so any difference observed would be due to experimental errors. Hence, increasing the likelihood of *Type I error* implies rejecting H_0 in favor of the alternative hypothesis, H_1; this means that it is assumed the existence of significant difference among the execution times. In an experimental design in which there is a group of treatments, the intent is to compare them in pairs in order to identify if there is a significant difference between them. The results of these comparisons are a set which is known as *family* (Howell 2010). When comparing only two treatments, the obtained result is due to an alpha value (e.g., α=0.05) that is the probability of obtaining a *Type I error* for this comparison. However, when evaluating a *family* in which the number of comparisons is greater than two, it is necessary to evaluate the probability of rejecting incorrectly, at least, one of the null hypotheses that comprise this *family* (Howell 2010).

In face of the problem exposed above, in this study we changed the protocol described in (Touati, Worms, and Briais 2013), so as to enable analyzing samples from multiple treatments with no FWER influence. Note that dealing with experimental plans composed of multiple treatments is a reality in many practical experimental studies.

Figure 1 shows the execution time samples comparison protocol hereby proposed. A first difference between our protocol proposal and the one presented in (Touati, Worms, and Briais 2013) is that our version adjusts the *p-values* (Glantz 2011) at the end of the comparisons, in order to mitigate the influence of the FWER problem, thus dealing with the problem of comparing multiple treatments. In both protocols, the execution time samples are considered statistically different when the assessment of the test's statistics yields a *p-value* below 0.05 (α = 5%).

In Figure 1, different execution time samples, X_{it}, are obtained by executing the treatments of interest. Based on a given significance level, α, the protocol uses the *Student's t-test* to assess whether the mean of X_{it} is higher than the mean of each one of the other treatments' samples. Note that *Student's t-test* requires that both tested samples follow a Gaussian distribution and have the same variance (Sheskin 2003). Therefore, these assumptions must be previously verified using, respectively, *Shapiro-Wilk test* (Shapiro and Wilk 1965) and *Fisher's F-test* (Sheskin 2003). If the samples do not have the same variance, then the protocol requires the use of *Welch's t-test* (Howell 2010). Unlike the method described in (Touati, Worms, and Briais 2013), if it is confirmed that the data do not follow a Gaussian distribution, our protocol applies the *Wilcoxon-Mann-Whitney test* instead of *Welch's t-test*. The *Wilcoxon-Mann-Whitney test* consists in ranking the observations and computing the sum of the ranks for each group. However, in order to apply this test, the assumption that the samples come from the same distribution must be met. In the approach hereby proposed, we use the *Kolmogorov-Smirnov test* (Gibbons and Chakraborti 2014) to determine whether this assumption is true or false. In the case of multiple comparisons, our protocol adopts the *Holm's test* (Glantz 2011) in order to keep the influence of the FWER problem under control. Thus, after obtaining all *p-values* of all comparisons, we sort the *p-values* smaller than α, in ascending order. Next, we apply the *Holm's test* to evaluate whether there is the presence of false positive.

The *Holm's test* is considered an improvement of the approach proposed by Bonferroni (Glantz 2011), being a less conservative adjustment procedure. Unlike the Bonferroni's test, the *Holm's test*

considers the previous comparisons to evaluate subsequent comparisons; this test is proposed to accept or reject a set of null hypotheses, ordered and starting with the lowest *p-value* (Glantz 2011). Thus, if there are k paired comparisons, it should sort k *p-values* in ascending order, where p_k is the lowest unadjusted *p-value*, j is the jth hypotesis test performed ($j \leq k$), and α_τ is the real significance level, so the *Holm's test* is given by iteration of $\alpha_j = \alpha_\tau / (k - j + 1)$. For a certain iteration, if the kth *p-value* is less than the adjusted *p-value*, α_j, the test rejects the null hypothesis and then starts the next iteration, comparing the next *p-value* in the sorted list. Otherwise, if the unadjusted *p-value* is higher than α_j, then the test should be stopped because it is assumed that the subsequent comparisons are not significant (Glantz 2011).

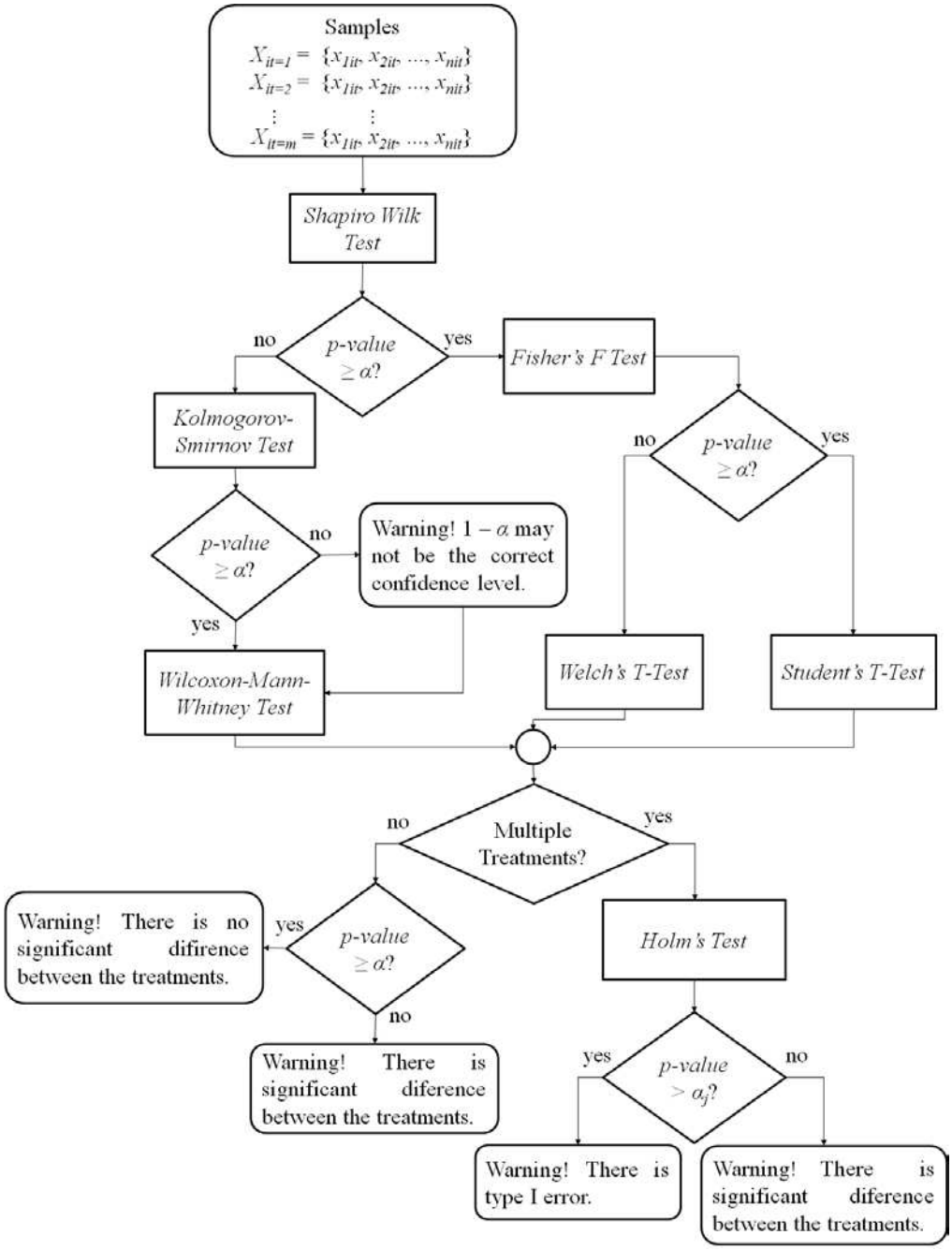

Figure 1: Proposed protocol to compare samples of execution times.

3.6 Testbed Environment and Instrumentation

For carry out our experiments we used a NUMA computer with four AMD Opteron™ Processor 6212 of 1.40 GHz and 8 cores each (32 cores in total), with three levels of cache and 64GB of RAM (see Figure 2). The operating system used was the Linux kernel version 3.11.10-7 (OpenSUSE 13.1).

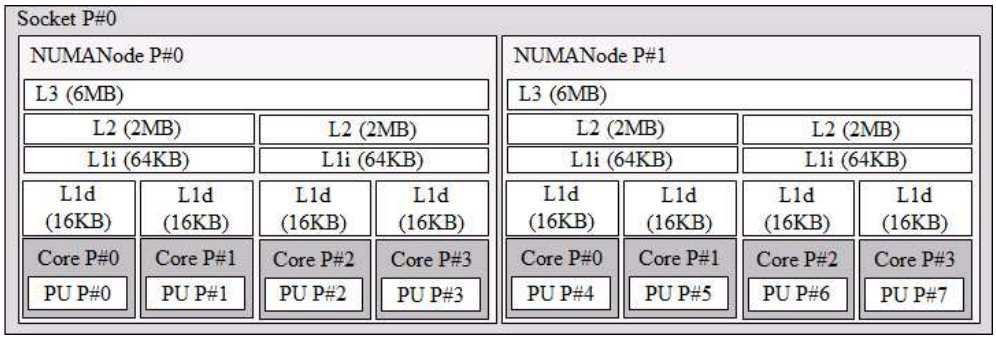

Figure 2: Topology of one of the four processors used.

4 RESULTS

4.1 Experiment #1

After obtaining the execution time samples for each treatment assessed in this experiment, we applied the protocol described in Section 3.5 (see Figure 1).

Firstly, by comparing the treatments in this experiment, it was observed that for the pairs RL3O2-RL5O2, RL3O2-RL5O3, and RL5O2-RL5O3, in 100% of cases the comparisons were performed through the *Wilcoxon-Mann-Whitney test*. For the pairs RL3O2-RL3O3, RL3O3-RL5O2, and RL3O3-RL5O3, the *Student's t-test* was applied in 10% of cases; no comparison used the *Welch's t-test*. Hence, we conclude that most of the samples showed no adherence to a Gaussian distribution, given that most of the comparisons (over 90%) were performed using the *Wilcoxon-Mann-Whitney test*. These evidences suggest that the factors *runlevel* and *compiler optimization* significantly influenced the variability observed in the execution times, which changed the execution time distributions in different treatments. Note that this result shows that not always the execution times follow a Gaussian distribution, which is not infrequently assumed by studies in the literature (e.g., Georges, Buytaert, and Eeckhout 2007; Mazouz, Touati, and Barthou 2010).

Table 3 shows the results of all paired comparisons among the treatments for all NPB benchmarks, before applying the *Holm's test*. It can be seen that the pairs of treatments RL3O2-RL3O3, RL3O2-RL5O3, RL3O3-RL5O2, and RL5O2-RL5O3 presented differences considered statistically significant in over 90% of the comparisons.

Table 3: Percentages of paired comparisons different statistically in Exp. #1.

	RL3O3	**RL5O2**	**RL5O3**
RL3O2	100%	50%	100%
RL3O3		90%	50%
RL5O2			90%

Table 4 shows the results for all paired comparisons after applying the *Holm's test*. The asterisk symbol (*) indicates the comparisons that had a reduction in their percentage of comparisons considered statistically different, with reference to Table 3, due to *Type I errors* detected during the *Holm's test* analysis. In general, we noticed that the set of comparisons was statistically significant and that only one

of the ten NPB benchmark programs, *SP*, presented a comparison result (RL3O2-RL3O3) with *Type I error* due to the FWER problem; this specific comparison represented 33.33% of the *SP*'s treatment comparisons considered statistically significant. The results after the *Holm's test* confirm the significant influence of factor *compiler optimization* (O) on the execution times of the NPB's benchmarks (≥ 90% of comparisons); the factor *runlevel* (RL) showed a lower influence (50% of comparisons).

Table 4: Percentages of paired comparisons different statistically in Exp. #1 with *Holm's test* analysis.

	RL3O3	**RL5O2**	**RL5O3**
RL3O2	90%*	50%	100%
RL3O3		90%	50%
RL5O2			90%

4.2 Experiment #2

The results of Experiment #2 showed that, in overall, no specific *environment variable size* stood out among the longest and shortest execution times, considering all evaluated programs. This was likely given that each program has a different behavior, which interferes in the results.

Initially, by comparing the treatments with the optimization factor in O2, it was observed that for the pairs 0-64, 0-1024, 64-128, 64-256, 64-512, 64-1024, and 64-2048, in 100% of cases the comparisons were performed through the *Wilcoxon-Mann-Whitney test*. For the pair 0-128, the *Student's t-test* was applied in 30% of cases, and in pairs 64-4096 and 2048-4096 the *Welch's t-test* was used in 10% of cases. For the optimization factor in O3, it was observed that for the pairs 0-128, 64-128 and 128-2048, the *Wilcoxon-Mann-Whitney test* was used in 100% of cases; for the pairs 256-2048, 256-4096, 512-1024 and 2048-4096 the *Student's t-test* was applied in 20% of cases, and for the pairs 128-4096, 512-4096 and 1024-4096 the *Welch's t-test* was used in 10% of cases. Hence, we conclude that most of the samples showed no adherence to a Gaussian distribution, given that most of the comparisons (over 70%) were performed using the *Wilcoxon-Mann-Whitney test*.

Table 5 shows the results of all paired comparisons before applying the *Holm's test*. We observe that over 20% of all comparisons were considered statistically significant different, for both levels of the optimization factor (O2 and O3). We also noted that there was difference in the percentage of the comparisons statistically different when we compared the results obtained with the optimization factor in levels O2 and O3. For example, when comparing the treatments with EVS varying from 0 to 2048 in O2, we observed that 20% of the comparisons showed statistically significant difference in the execution times. On the other hand, when the optimization factor was changed to O3 we identified that this percentage increased to 70% (see Table 5).

Table 6 shows the results of all paired comparisons after applying the *Holm's test*. Analyzing the results obtained with and without the *Holm's test* applied to the comparisons, we note that except for comparisons 0-128 and 256-1024 with the optimization factor in O2, and comparisons 128-256, 0-1024, 256-1024, 512-1024 and 64-2048 with the optimization factor in O3, all other comparisons presented false positive (presence of *Type I error*). Specifically in the treatments where the optimization factor was set to O2, we found that except for the results obtained for the NPB programs *GC* and *IS*, all other results would be affected by the FWER problem without applying our proposed control. For example, in case of the *BT* program, 47.05% of its comparisons would be considered statistically different incorrectly; the other programs showed the following percentages of false positives: *DC* (69.23%), *EP* (33.33%), *FT* (45.45%), *LU* (27.78%), *MG* (17.39%), *SP* (77.78%), and *UA* (100.00%). Regarding the treatments where the optimization factor was set to O3, except for the *EP* benchmark program, all other NPB programs would be affected by the FWER problem without our proposed control. The *BT* program would have 30% of its comparisons considered statistically different containing *Type I errors*, so as the others programs:

CG (7.41%), *DC* (100%), *FT* (54.54%), *IS* (35%), *LU* (33.33%), *MG* (21.05%), *SP* (80%), and *UA* (77.78%).

Table 5: Percentages of paired comparisons different statistically in Exp. #2.

	O2 Optimization						
EVS	**64**	**128**	**256**	**512**	**1024**	**2048**	**4096**
0	50%	40%	60%	70%	50%	20%	50%
64		70%	60%	40%	70%	60%	70%
128			40%	30%	50%	40%	50%
256				40%	40%	30%	60%
512					50%	50%	50%
1024						50%	70%
2048							50%
	O3 Optimization						
EVS	**64**	**128**	**256**	**512**	**1024**	**2048**	**4096**
0	60%	50%	50%	40%	30%	70%	40%
64		60%	50%	60%	70%	60%	60%
128			60%	60%	70%	70%	60%
256				60%	20%	90%	40%
512					40%	60%	50%
1024						60%	40%
2048							60%

Table 6: Percentages of paired comparisons different statistically in Exp. #2 with *Holm's test* analysis.

	O2 Optimization						
EVS	**64**	**128**	**256**	**512**	**1024**	**2048**	**4096**
0	30%*	40%	20%*	50%*	40%*	10%*	30%*
64		40%*	40%*	30%*	60%*	30%*	60%*
128			20%*	20%*	30%*	20%*	40%*
256				30%*	40%	20%*	40%*
512					20%*	30%*	40%*
1024						40%*	40%*
2048							40%*
	O3 Optimization						
EVS	**64**	**128**	**256**	**512**	**1024**	**2048**	**4096**
0	60%	40%*	40%*	10%*	30%	40%*	20%*
64		50%*	40%*	40%*	40%*	60%	40%*
128			60%	40%*	50%*	60%*	30%*
256				40%*	20%	60%*	30%*
512					40%	50%*	40%*
1024						40%*	30%*
2048							50%*

In summary, this experiment shows that the same benchmark programs running with different sizes of one or more OS *environment variables* may result in statistically significant different execution times, which certainly affect experimental analyses if this influence is not adequately controlled. It is important

to highlight that it is not uncommon for studies involving computing experiments not to consider this type of influence in their planning or result analyses. Note that changes in the size of OS *environment variables* can be done intentionally by the experimenter or even with no user intervention (e.g., by operating system programs or services). In both situations, not appropriately considering the influence of this factor on the execution times may cause erroneous conclusion, leading to the belief that the differences observed in the execution times are consequences of one or more factors being tested (e.g., a new algorithm), while there may actually be the effect of uncontrolled external factors. The results also showed that our proposed approach was adequate to prevent the influence of *Type I errors*, caused by the FWER problem, which could affect a significant percentage of the analyzed results.

4.3 Experiment #3

Based on the results of this experiment, we analyzed the comparisons and observed that for the treatment pairs SA2TH-CP2TH, SA4TH-CP2TH, CP2TH-CP4TH, and CP2TH-CP6TH the *Wilcoxon-Mann-Whitney test* was used in 100% of cases, which means their execution time samples did not follow a Gaussian distribution. For the pair SA2TH-SC2TH the *Student's t-test* was used in 30% of cases; the *Welch's t-test* was used in 40% of the comparisons in the treatment pairs SA2TH-CP4TH, SA6TH-CP4TH, SA6TH-CP6TH, CP4TH-SC2TH and CP4TH-SC6TH. The results indicate that the factors *number of threads* and *thread allocation strategy* had significant influence on changing the distribution of the execution times.

Table 7 shows the results of all paired comparisons before applying the *Holm's test*. In assessing the difference between the treatments, the results show that all comparisons should be considered statistically different in over 60%. Table 8 shows the results after applying the *Holm's test* for the all comparisons. The results indicate that the *MG* program was affected by the FWER problem. The detected false positive, without the proper control, would represent 3.03% of the *MG*'s treatment comparisons considered different statistically.

Table 7: Percentages of paired comparisons different statistically in Exp. #3.

	SA 4TH	**SA 6TH**	**CP 2TH**	**CP 4TH**	**CP 6TH**	**SC 2TH**	**SC 4TH**	**SC 6TH**
SA2TH	100%	100%	100%	90%	100%	60%	100%	100%
SA4TH		100%	100%	90%	90%	100%	100%	90%
SA6TH			100%	100%	90%	100%	100%	90%
CP2TH				100%	100%	90%	100%	100%
CP4TH					100%	100%	100%	100%
CP6TH						100%	100%	90%
SC2TH							100%	100%
SC4TH								100%

5 CONCLUSION

Computing experiments are crucial for scientific research. Nonetheless, the accuracy of experimental results greatly depends on the rigor applied to the design of the experiments, as well as their execution and output analyses. In this paper, we present empirical evidences that expose the importance of taking into account and handling the variability in execution times of computer programs, which are caused mainly by environmental factors, particularly the ones related to the operating systems. Neglecting such influences during the analysis of computing experiments put in risk the correct understanding of their results.

Table 8: Percentages of paired comparisons different statistically in Exp. #3 with *Holm's test* analysis.

	SA 4TH	**SA 6TH**	**CP 2TH**	**CP 4TH**	**CP 6TH**	**SC 2TH**	**SC 4TH**	**SC 6TH**
SA2TH	100%	100%	100%	90%	100%	60%	100%	100%
SA4TH		100%	100%	90%	90%	100%	100%	90%
SA6TH			100%	100%	90%	100%	100%	90%
CP2TH				100%	100%	90%	100%	100%
CP4TH					100%	90%*	100%	100%
CP6TH						100%	100%	90%
SC2TH							100%	100%
SC4TH								100%

The experimental findings we present confirm that the size of operating system *environment variables* may significantly impact program execution times. The same was observed for the *runlevel* and the *thread affinity* factors, which are regularly not considered by experimenters in computing experiments.

Another important finding is that not always do the execution time samples follow a Gaussian distribution; this is an assumption not rarely found in related works in the literature. Furthermore, we experimentally show that the *familywise error rate* problem is present in multiple comparisons of execution times in computing experiments and also may influence a significant high number of treatment comparisons; in some cases we observed that 100% of the results could be affected by this problem without the proper control.

Finally, our proposed changes to the protocol originally introduced in Touati, Worms, and Briais (2013) added the necessary support for the proper analysis of execution times obtained from multiple treatments, controlling the overall probability of observing one or more *Type I errors*. This approach keeps the effects of FWER problem under control for this category of experimental data.

Two ongoing related research works are under development. One is focused on the investigation of different procedures, widely adopted in the literature, to measure the execution times of computer programs. Our goal is to evaluate their accuracy and also possible influences on the execution time variability. The second work is creating a software platform to help experimenters to design and automatically execute computing experiments based on the DOE approach. This platform will support the proposed protocol presented in this work for the analysis of execution times obtained from experimental designs based on multiple treatments.

ACKNOWLEDGMENTS

This work was supported partially by the Brazilian research agencies CNPq, CAPES and FAPEMIG.

REFERENCES

Georges, A., D. Buytaert, and L. Eeckhout. 2007. "Statistically Rigorous Java Performance Evaluation." *ACM SIGPLAN Notices* 42: 57-76.

Gibbons, J. D., and S. Chakraborti. 2014. *Nonparametric Statistical Inference*. 4th ed. New York: Marcel Dekker, Inc.

Glantz, S. A. 2011. *Primer of Biostatistics*. 7th ed. New York: McGraw-Hill, Inc.

GNU. 2014. "GNU Libgomp." Accessed July 20. https://gcc.gnu.org/onlinedocs/libgomp/.

Howell, D. C. 2010. *Statistical Methods For Psychology*. 7th ed. Belmont: Cengage Wadsworth.

INTEL. 2014. "ICC, Intel® C and C++ Compilers." Accessed July 20. https://software.intel.com/en-us/c-compilers/.

Jain, R. 1991. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. John Wiliey.Kerrisk, M. 2010. *The Linux Programming Interface*. 1st ed. San Francisco: No Starch Press.

Lilja, D. J. 2005. *Measuring Computer Performance: A Practitioner's Guide*. 1st ed. Cambridge: Cambridge University Press.

Mazouz, A., S.-A.-A. Touati, and D. Barthou. 2010. "Study of Variations of Native Program Execution Times on Multi-Core Architectures." In *Proceedings of the 4th International Conference on Complex, Intelligent and Software Intensive Systems*, 919–924. Washington, DC: IEEE Computer Society.

Mazouz, A., S.-A.-A. Touati, and D. Barthou. 2011a. "Analysing the Variability of openMP Programs Performances on Multicore Architectures." In *Proceedings of the 4th Workshop on Programmability Issues for Heterogeneous Multicores*, 1–14.

Mazouz, A., S.-A.-A. Touati, and D. Barthou. 2011b. "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 & Simulation*, 273–279. IEEE.

Montgomery, D. C. 2000. *Design and Analysis of Experiments*. 5th ed. New York: John Wiliey & Sons, Inc.

Mytkowicz, T., A. Diwan, M. Hauswirth, and P. F. Sweeney. 2009. "Producing Wrong Data without Doing Anything Obviously Wrong!" *ACM SIGPLAN Notices* 44: 265-276.

NASA Advanced Supercomputing Divison. 2014. "NAS Parallel Benchmarks." Accessed July 20. http://www.nas.nasa.gov/publications/npb.html.

Pusukuri, K. K., R. Gupta, and L. N. Bhuyan. 2012. "Thread Tranquilizer: Dynamically Reducing Peformance Variation." *ACM Transactions on Architecture and Code Optimization* 8: 1–21.

Shacham, H., M. Page, B. Pfaff, E.-J. Goh, N. Modadugu, and D. Boneh. 2004. "On the Effectiveness of Address-Space Randomization." In *Proceedings of the 11th ACM Conference on Computer and Communications Security*, 298-307. New York, New York: ACM.

Shapiro, S. S., and M. B. Wilk. 1965. "An Analysis of Variance Test for Normality (Complete Samples)." *Biometrika* 52: 591–611.

Sheskin, D. J. 2003. *Handbook of Parametric and Nonparametric Statiscal Procedures*. 3th ed. Boca Raton: CRC Press.

Stallman, R. M, and The GCC Developer Comunity. 2012. "Using the GNU Compiler Collection." *Development*. Boston: GNU Press. https://gcc.gnu.org/onlinedocs/gcc-4.7.2/gcc/.

Touati, S.-A.-A., J. Worms, and S. Briais. 2013. "The Speedup-Test: A Statistical Methodology for Programme Speedup Analysis and Computation." *Concurrency and Computation: Practice and Experience* 25: 1410–1426.

Vicente, E., and R. Matias Jr. 2013. "Modeling and Simulating the Effects of OS Jitter." In *Proceedings of the 2013 Winter Simulation Conference: Simulation*, edited by A. Tolk, S.-H. Kim, R. Pasupathy, M. Kuhl, and R. Hill, 2151–2162. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

AUTHOR BIOGRAPHIES

PAULO EDUARDO NOGUEIRA is a graduate student in Computer Science at Federal University of Uberlandia, Brazil. His e-mail address is paulo.nogueira@ifgoiano.edu.br.

RIVALINO MATIAS JR. is an Associate Professor in the School of Computer Science at Federal University of Uberlandia, Brazil. He holds M.S. and Ph.D. degrees in Computer Science and Industrial and Systems Engineering, respectively, from Federal University of Santa Catarina, Brazil. His e-mail and web addresses are rivalino@ufu.br and http://hpdcs.facom.ufu.br/.