Estudo Quantitativo da Variabilidade nos Tempos de Execução de Experimentos Computacionais

Paulo Eduardo Nogueira

Instituto Federal Goiano
Morrinhos, Brasil
paulo.nogueira@ifgoiano.edu.br

Rivalino Matias Jr.
Faculdade de Computação
Universidade Federal de Uberlândia
Uberlândia, Brasil
rivalino@fc.ufu.br

***Resumo*— Muitos trabalhos de pesquisa experimental em computação dependem da correta mensuração dos tempos de execução de programas de computador. Observa-se que nem todos consideram que repetidas execuções do mesmo programa podem produzir tempos de execução significativamente diferentes em termos estatísticos. A falta de rigor na análise dos tempos de execução de programas de computador tem sido objeto de vários estudos na literatura. Neste trabalho, primeiramente foram reproduzidos experimentos da literatura, com o propósito de analisar as propriedades estatísticas dos seus resultados em termos de tempos de execução, bem como avaliar os efeitos de diferentes fontes de variabilidade nos tempos de execução. Em especial, foram consideradas fontes relacionadas com o sistema operacional. Também, foi proposto um protocolo para sistematizar a comparação de tempos de execução de programas, para identificar diferenças estatisticamente significativas em amostras obtidas de experimentos com múltiplos tratamentos (cenários de teste).**

Palavras-chave*— *tempos de execução, variabilidade, sistemas operacionais

I. INTRODUÇÃO

Em engenharia de sistemas de computação, muitos trabalhos de pesquisa dependem da análise dos tempos de execução de programas de computador [1]. Por exemplo, ao avaliar experimentalmente o desempenho de dois algoritmos em um dado sistema, o pesquisador compara os tempos de execução de suas implementações. Nestes casos, é essencial reproduzir os resultados experimentais de forma consistente, para se obter confiança estatística nos resultados.

Em sistemas computacionais, nem sempre execuções sucessivas de um mesmo programa produzem os mesmos tempos de execução. De fato, o que se observa na prática é que, muitas vezes, ocorre uma variação significativa nos tempos de execução do programa, durante sucessivas execuções. Em pesquisa experimental, essa variação é conhecida como erro experimental e é causada por fatores não controláveis [2]. Em experimentos computacionais esses fatores podem estar relacionados com o *hardware* ou *software*, tais como a influência de mecanismos de interrupção, da arquitetura da memória *cache*, e diferentes interferências provocadas por rotinas do sistema operacional (SO) conhecidas como *OS Jitter* [3], entre outros fatores. Neste trabalho estudaremos os fatores de software relacionados com o SO.

A natureza estocástica das variações provocadas pelos fatores citados anteriormente faz com que sua magnitude seja de difícil predição. Como consequência, vários trabalhos experimentais, especialmente aqueles que dependem de correta análise dos tempos de execução de programas, falham ao tratar esse problema. Não é incomum encontrar trabalhos de pesquisa que reportam resultados de experimentos computacionais baseados em uma única execução do experimento. Por causa do problema de variação anteriormente citado, fica claro que se basear em uma única execução não é uma abordagem confiável, uma vez que o único tempo de execução observado pode desviar-se, significantemente, dos valores mais frequentes. A falta de rigor em lidar com erros experimentais em experimentos computacionais tem sido objeto de investigação em vários trabalhos da literatura (ex. [1], [3], [4], [5], [6], [7], [8] e [9]).

Neste artigo, apresenta-se um estudo experimental sobre a variação nos tempos de execução de programas de computador. Foram consideradas diferentes fontes de variação de tempos de execução relacionadas com o SO, as quais são: nível de *Runlevel* [3], tamanho de variáveis de ambiente [5] e estratégias de alocação de *threads* [7]. Complementarmente, realizou-se um melhoramento do método de análise de tempos de execução proposto em [1], oferecendo maior precisão na análise de dados de múltiplos cenários experimentais (tratamentos). O restante desse artigo está organizado como segue. A Seção II apresenta os trabalhos relacionados. A Seção III apresenta os métodos, os materiais utilizados e o estudo experimental. A Seção IV discute os resultados obtidos e a Seção V apresenta as conclusões.

II. TRABALHOS RELACIONADOS

De acordo com Touati *et al*. [1], diferentes áreas de pesquisa em ciência da computação possuem dificuldades em reproduzir resultados experimentais quando se trata de tempo de execução de programas. Os autores propõem uma abordagem estatística para comparar os tempos de execução de duas versões do mesmo programa com a mesma entrada. A proposta apresenta resultados mais precisos do que outras abordagens descritas na literatura (ex. [4] e [10]).

Georges *et al*. [4] revisaram vários métodos para avaliar o tempo de execução de aplicações Java, usando métricas propostas em diferentes trabalhos. Ao usar uma análise estatística mais rigorosa, os resultados dos trabalhos anteriores

não foram confirmados. Também, foram avaliados métodos predominantes na literatura e verificou-se que seus resultados poderiam estar errados em até 16% dos casos.

Em [5], [6] e [7] os autores observaram que diferentes fatores do sistema operacional causam importante variabilidade nos tempos de execução de programas, afetando assim a reprodutibilidade dos resultados. Mytkowicz *et al*. [5] alterou a ordem de ligação do código (*linking order*) e também o tamanho de variáveis de ambiente do UNIX, para avaliar suas influências no tempo de execução de programas. Eles observaram que a primeira alteração afeta significantemente a magnitude das variações do tempo de execução. Também, constataram que aumentando o tamanho de variáveis de ambiente ocorreu degradação nos tempos de execução dos programas analisados. Mazouz *et al*. [6] avaliaram a variabilidade no tempo de execução de aplicações sequenciais e paralelas. Os resultados demonstraram que aplicações paralelas são mais suscetíveis a tempos de execução variáveis do que aplicações não paralelas. Mazouz *et al*. [7] avaliaram o uso de diferentes estratégias de *thread affinity* e encontraram uma notável variabilidade nos tempos de execução quando há migração de *threads*, corroborando os resultados de [8].

Pusukuri *et al*. [9] mostram que tempos de execução em sistemas *multicore* são altamente sensíveis às políticas de gerência de processador do sistema operacional. Eles demonstram que uma alteração em tais políticas pode afetar a variabilidade dos tempos de execução em até 98%.

Como se pode observar, há muitos fatores relacionados ao sistema operacional que afetam resultados experimentais em termos de tempo de execução. Diferente dos trabalhos anteriores, neste estudo confirmou-se tais influências por meio de um método experimental rigoroso, utilizando experimentos controlados e repetidos. A análise da variabilidade dos tempos de execução se deu com um protocolo que considera múltiplos tratamentos e o problema *familywise error rate*, diferente da abordagem usada em [1] que não considera tal problema.

III. Estudo Experimental

A. Método

Neste trabalho foi adotado o método estatístico DOE (*Design of Experiments*) [2] para o planejamento e execução dos experimentos, bem como a análise dos seus resultados. Esse método requer modificações controladas dos fatores sendo estudados, com o propósito de observar os efeitos dessas alterações sobre a variável resposta. A variável resposta de interesse neste estudo foi o tempo de execução dos programas selecionados para cada cenário de teste (tratamento). Cada fator (fonte de variabilidade dos tempos de execução) assume diferentes níveis de operação. Configura-se como um tipo de tratamento uma dada combinação de fatores e níveis [2]. Para a definição dos tratamentos foi utilizado o método da matriz de sinais [11] configurada de acordo com a ordem de Yates [2]. Para evitar que a execução de um tratamento influenciasse nos resultados da execução do próximo tratamento, a cada novo tratamento o SO foi reinicializado.

Após a realização de cada experimento, os tempos de execução observados para um tratamento são comparados com os tempos dos demais tratamentos. Se existir diferença significativa, aqui definida por meio de teste estatístico de comparação de amostras, ao nível de significância de 5%, isso significa que um ou mais fatores sendo analisados naqueles tratamentos têm influência importante nos tempos de execução dos programas. Isso indica que as condições experimentais destes tratamentos devem ser cuidadosamente consideradas ao se analisar tempos de execução de programas executados nas mesmas condições, sob pena de se cometer erros na interpretação dos seus resultados. Os tempos de execução dos programas foram obtidos através do programa *time* [12].

B. Planejamento do Experimento #1

Esse experimento teve por objetivo avaliar o efeito do nível de *Runlevel* e da *Otimização* do compilador sobre o tempo de execução de programas do *benchmark* NPB 3.3.1 [13]. O NPB é um conjunto de programas projetado para avaliar o desempenho de computadores paralelos, sendo baseado em aplicações de CFD (*computational fluid dynamics*). A Tabela I sumariza os fatores e níveis utilizados neste experimento.

TABELA I. Fatores e Níveis Avaliados

		Nível (-)	**Nível (+)**
Fatores	*Runlevel* (RL)	5	3
	Otimização (O)	O2	O3

No nível (-) o fator *Runlevel* assume o valor 5, o que indica um número maior de processos administrativos (serviços) do SO executando em segundo plano (*background*); o nível (+) configura este fator para 3, o que resulta em um número reduzido de processos administrativos. Variando esse fator, busca-se verificar o efeito dos processos administrativos do SO no tempo de execução do NPB. Já o fator *Otimização* refere-se ao nível de otimização aplicada a cada programa do NPB durante sua compilação. Foi usado o compilador *gcc* versão 4.7.2 [14]. No nível (-), este compilador utilizou a otimização O2 e no nível (+) a otimização O3. A opção O3 indica uma otimização mais "agressiva" do que a O2, resultando no aumento no tamanho do código executável.

Neste experimento, todos os programas do NPB foram executados de forma concomitante. Cada tratamento foi replicado 31 vezes, sendo que a primeira replicação foi descartada visando reduzir a influência do *buffer-cache* de disco que existe na primeira execução. As replicações objetivam obter uma amostra de tamanho suficiente que permita obter uma estimativa adequada dos erros experimentais, ajudando a determinar se as diferenças entre os tratamentos são estatisticamente significativas ou não.

C. Planejamento do Experimento #2

Este experimento reproduziu o experimento realizado em [5], cujo objetivo foi avaliar o efeito do tamanho de variáveis de ambiente do SO sobre o tempo de execução de programas. Este é um fator pouco considerado no planejamento e análise de experimentos computacionais. Em [5], os autores relatam que alterações no tamanho destas variáveis afetam o

alinhamento da pilha do programa e, por conseguinte, o alinhamento das estruturas alocadas na *heap*, influenciando no tempo de execução dos programas.

Diferente do Exp. #1, onde os programas do NPB foram executados concomitantemente, nesse experimento cada programa do NPB foi executado individualmente, seis vezes para cada tamanho diferente de variável de ambiente. Pela mesma razão do experimento anterior, a primeira execução para cada tamanho de variável foi descartada. A variável de ambiente utilizada foi criada para esse propósito, recebendo um valor *string* cujo tamanho variou de 0 até 4096 bytes, aumentando 64 bytes a cada tratamento. O recurso do Linux de randomização do endereço inicial da pilha de memória foi desabilitado, com a intenção de se avaliar isoladamente os efeitos dos diferentes tamanhos da variável de ambiente sobre a pilha dos programas. Todos os programas do NPB foram compilados com as otimizações O2 e O3, assim como no Exp. #1. No caso do *Runlevel*, adotou-se o valor 1 a fim de ter o menor número possível de processos administrativos executando durante os tratamentos, reduzindo assim sua influência nos resultados.

D. Planejamento do Experimento #3

O experimento #3 avaliou o fator *thread affinity*, o qual representa a estratégia de alocação de múltiplas *threads* nos processadores (ou núcleos). Esse experimento reproduz o experimento realizado em [7], objetivando analisar a influência da alocação de *threads* na variabilidade dos tempos de execução dos programas. Em [7] foi utilizado o compilador *icc* [15] que define as seguintes estratégias: *no affinity*, *compact* e *scatter*. Na primeira o SO é livre para alocar as *threads* nos processadores de acordo com sua disponibilidade. Na segunda, o SO é instruído a alocar as *threads* em cada núcleo de modo sequencial, tão próximas quanto possível, para aumentar as chances de compartilhamento da *cache* de processador. Na terceira, o SO é instruído a distribuir as *threads* pelos núcleos o mais uniformemente possível.

Neste trabalho, a reprodução destas três estratégias de afinidade de *threads* foi possível utilizando as funcionalidades implementadas pela OpenMP (*libgomp*) [16] e *gcc*. Neste caso, a escolha da estratégia é feita pela variável de ambiente *GOMP_CPU_AFFINITY*. A configuração do *gcc+libgomp* compatível com a *icc-compact* é *GOMP_CPU_AFFINITY*=0-7. Para implementar a *icc-scatter*, utilizou-se *GOMP_CPU_AFFINITY*=0 4 2 6 1 5 3 7. Cada programa do NPB foi executado 31 vezes em *Runlevel* 1, sendo que a primeira execução foi descartada como nos demais experimentos. A Tabela II apresenta os níveis e fatores para esse experimento. Uma vez que cada fator foi avaliado em três níveis, foram realizados 3^2 tratamentos.

TABELA II. FATORES E NÍVEIS AVALIADOS

		Níveis
Fatores	*Número de Threads* (NT)	2, 4 e 6
	Estratégia de Affinity (EA)	*No Affinity, Compact* e *Scatter*

E. Protocolo para Análise dos Tempos de Execução

Ao estudar a abordagem descrita em [1], a qual avalia a significância estatística da variação de duas amostras de tempos de execução, verificou-se que seu método é inadequado para tratar casos onde as amostras são obtidas de múltiplos tratamentos, ou seja, comparar amostras de diferentes cenários experimentais. Isso ocorre porque os testes usados em [1] para avaliar a significância estatística das diferenças nos tempos de execução, pela média e mediana, testes *t* de *Student* e *Wilcoxon-Mann-Whitney* [17] respectivamente, são indicados para comparar apenas duas amostras.

O emprego destes testes para comparar múltiplos tratamentos leva ao problema conhecido como *familywise error rate* [18], o que aumenta a probabilidade de erro do tipo I [17], ou seja, rejeitar a hipótese nula (H_0) dado que ela é verdadeira. Nos testes supracitados, H_0 indica que não existe diferença significativa nos tempos de execução e a diferença observada é devido ao erro experimental, portanto, aumentar a chance do erro tipo I implica em rejeitar H_0 em favor da hipótese alternativa, a qual se refere a existência de diferença nos tempos de execução [18]. Por exemplo, ao avaliar os efeitos de diferentes algoritmos de escalonamento, em comparação com o algoritmo atual de um dado SO, observa-se que a média dos tempos de execução do programa de teste em um dado tratamento foi inferior à média dos tempos nos demais, o que sugere que neste tratamento houve redução no tempo de execução. Contudo, apesar dos dados apontarem em tal direção, esta conclusão tem maior chance de estar errada devido ao aumento do erro do tipo I, o qual ocorre por não tratar adequadamente a variabilidade inerente aos tempos de execução observados neste caso.

Face ao problema acima descrito, neste trabalho modificou-se o método proposto em [1], de forma a permitir analisar amostras de múltiplos tratamentos, tal como é realizado em diversos estudos experimentais, sem o problema *familywise error rate*. A Fig. 1 mostra o protocolo de comparação de tempos de execução aqui proposto. A diferença deste para o método apresentado em [1] é o fato de se realizar uma correção de *p-values* [19] ao final das comparações, de modo a se identificar se a rejeição de H_0 está correta, tratando assim o problema da comparação de múltiplos tratamentos. Neste protocolo, as amostras de tempos de execução são consideradas estatisticamente diferentes quando a avaliação da estatística do teste apresentar um *p-value* menor do que 0,05 ($\alpha = 5\%$).

Na Fig. 1, diferentes amostras de tempos de execução, X_1 até X_n, são obtidas executando os tratamentos de interesse. Baseado em um dado nível de significância, α, o protocolo proposto utiliza o teste *t* de *Student* para avaliar se a média de X_1 é maior do que a média das outras *n* amostras. Note que o teste *t* de *Student* requer que ambas as amostras sigam uma distribuição Gaussiana e possuam a mesma variância. Assim, essas suposições devem ser previamente avaliadas utilizando, respectivamente, os testes de *Shapiro Wilk* [20] e *F* de *Fisher* [17]. Se as amostras não possuem a mesma variância, então o protocolo requer o uso do teste *t* de *Welch* [18]. Diferente do método descrito em [1], ao se confirmar que os dados não seguem uma distribuição Gaussiana, o protocolo proposto

requer que seja utilizado um teste de *Wilcoxon-Mann-Whitney*, e não um teste de *Welch*.

O teste de *Wilcoxon-Mann-Whitney* consiste em ordenar (*ranks*) as observações e computar a soma dos *ranks* de cada grupo. No entanto, para aplicar esse teste é preciso satisfazer a suposição de que as amostras são provenientes de uma mesma distribuição. Na abordagem aqui proposta o teste de *Kolmogorov-Smirnov* [1] é usado para determinar se esta suposição é verdadeira ou falsa. No caso de múltiplas comparações, o protocolo proposto indica a realização do teste de *Holm* [19] para avaliar se as rejeições de H_0 estão corretas. O teste de *Holm* calcula um *p-value* ajustado a partir do *p-value* obtido da aplicação do teste de significância; se *p-value* > *p-value* ajustado, então deve-se rejeitar o teste uma vez que há a presença do erro do tipo I.

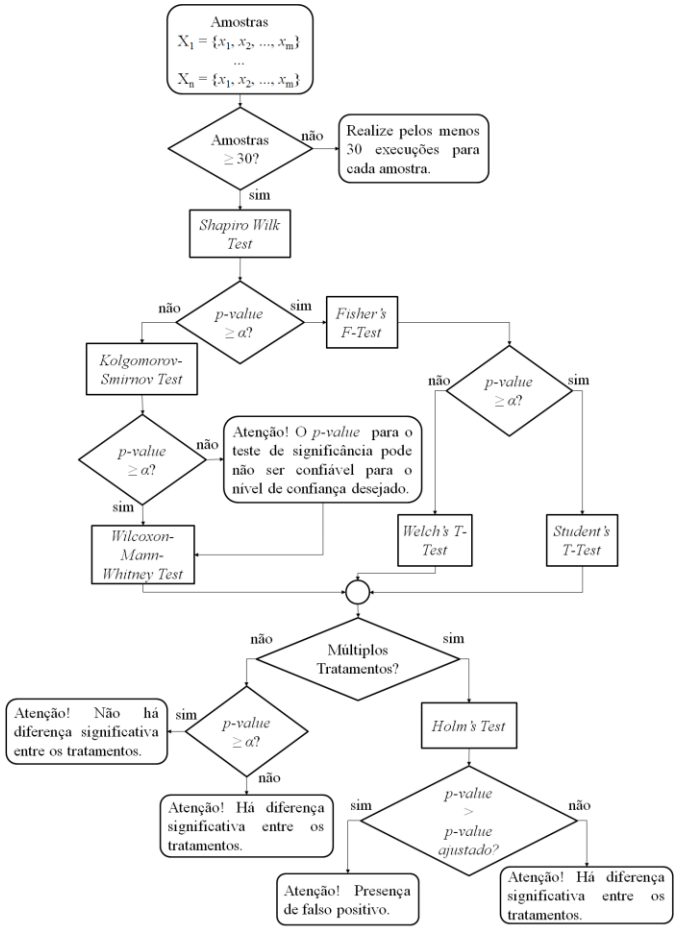

Fig. 1. Protocolo para comparação das amostras de tempo de execução.

F. Ambiente de Teste e Instrumentação

Dois ambientes de teste foram utilizados. Os experimentos #1 e #2 foram realizados em um computador com processador *Intel I5* 3570 *Quad-core* de 1,60 GHz, com três níveis de *cache* e 8 GB de memória RAM (ver Fig. 2). O sistema operacional adotado foi o Linux *kernel* 3.7.10-1.1 (*OpenSuse* 12.3).

O experimento #3, por explorar um nível maior de paralelismo utilizando *threads*, foi realizado em outro computador composto de 64GB de RAM e quatro processadores AMD Opteron 6212 de 1,40 GHz, tendo cada processor 8 cores e três níveis de *cache* (ver Fig. 3). O sistema operacional adotado foi o Linux *kernel* 2.6.32.28 (*OpenSuse* 13.1).

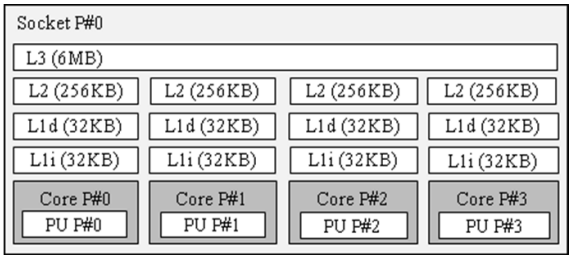

Fig. 2. Topologia do processador utilizado nos experimentos #1 e #2.

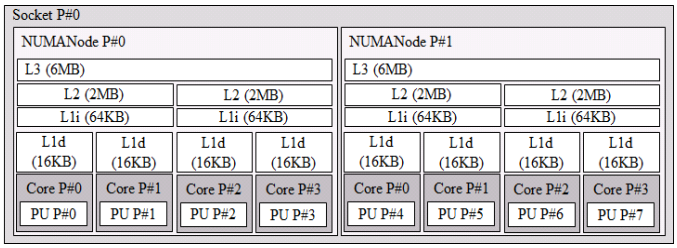

Fig. 3. Topologia de um dos quatro processadores usados no experimento #3.

IV. Resultados

A. Experimento #1

Após obter as amostras de tempos de execução para cada tratamento avaliado neste experimento, aplicou-se o protocolo de análise descrito na Seção III.E (ver Fig. 1).

Primeiramente, observou-se que dos dez programas do NPB, sete não apresentaram distribuição Gaussiana em pelo menos um dos tratamentos (ver Tabela III). O mesmo ocorre quando se avalia o tempo total do *benchmark*, ou seja, o tempo observado ao final da execução de todos os dez programas do NPB. A marcação "*X*" na Tabela III indica que os dados da amostra em determinado tratamento não seguem uma distribuição Gaussiana e a marcação "*O*" indica que os dados seguem esta distribuição.

TABELA III. Teste de Normalidade para os Tempos de Execução dos Programas do NPB

		RL3O2	RL3O3	RL5O2	RL5O3
Programas	BT	*O*	*O*	*X*	*O*
	CG	*O*	*O*	*O*	*O*
	DC	*O*	*X*	*O*	*O*
	EP	*X*	*X*	*X*	*O*
	FT	*O*	*O*	*X*	*O*
	IS	*O*	*O*	*O*	*O*
	LU	*X*	*O*	*O*	*O*
	MG	*X*	*O*	*O*	*X*
	SP	*O*	*X*	*O*	*O*
	UA	*O*	*O*	*O*	*O*
	Tempo Total	*O*	*X*	*O*	*O*

Estas evidências sugerem duas coisas. A primeira que o *Runlevel* e o tipo de *Otimização* do compilador influenciam, de forma significativa, a variabilidade dos tempos de execução,

alterando as distribuições dos tempos em diferentes tratamentos. A segunda, que nem sempre os tempos de execução seguem uma distribuição Gaussiana, o que às vezes é pressuposto em alguns trabalhos (ex. [4] e [6]).

Importante salientar que a maioria dos programas (7 de 10) obtiveram menor variância nos tempos de execução nos tratamentos em *Runlevel* 3 quando comparado com *Runlevel* 5, o que demonstra a influência do nível deste fator na variabilidade dos tempos de execução. Além disso, observa-se que os programas *EP*, *IS* e *MG*, não apresentaram diferença significativa em nenhum dos tratamentos, ou seja, estes dois fatores (RL e O) não afetam, de forma significativa, a execução desses programas tal como nos demais. A Tabela IV mostra os resultados de todas as comparações pareadas entre os tratamentos, para todos os programas do NPB. Observa-se que os pares RL3O2-RL3O3, RL3O2-RL5O3, RL3O3-RL5O2 e RL5O2-RL5O3 apresentaram diferença significativa em mais de 25% das comparações. Isto demonstra a influência do fator O sobre os tempos de execução dos programas. No caso do fator RL, foi observado que essas comparações estão abaixo dos 20%. Ao aplicar o teste de *Holm* constatou-se que todos os resultados são estatisticamente significativos, ao nível de 5%, e não foi detectado erro ao se rejeitar H_0.

TABELA IV. PERCENTUAL DE COMPARAÇÕES PAREADAS ESTATISTICAMENTE SIGNIFICANTES

	RL3O3	**RL5O2**	**RL5O3**
RL3O2	27,27%	18,18%	27,27%
RL3O3		36,36%	18,18%
RL5O2			63,64%

B. Experimento #2

Ao analisar os resultados do experimento #2, de forma geral nenhum tamanho específico de variável se destacou entre os maiores ou menores tempos de execução para todos os programas. Isso já era esperado, pois cada programa tem um comportamento diferente, o qual interfere nesse resultado.

De forma individual, em relação ao fator de ganho de cada programa, alguns padrões foram observados. No caso do programa *DC*, observou-se claramente que ganhos de desempenho ocorreram para sete tamanhos da variável de ambiente. No caso do programa *CG*, o fator de ganho e a variabilidade nos tempos de execução diminuiram, consistentemente, enquanto o tamanho da variável de ambiente aumentava, como pode ser observado na Fig. 4. Neste figura, cada ponto do gráfico representa a média do fator de ganho calculado para o respectivo tamanho de variável. Esses resultados corroboram os achados de [5], demonstrando que o tamanho da variável de ambiente afeta o tempo de execução dos programas. Importante salientar que não é incomum trabalhos que envolvem experimentos computacionais não considerarem esse tipo de influência no seu planejamento ou análise dos resultados.

Em resumo, um mesmo programa executando em um mesmo computador, porém com diferentes tamanhos de uma ou mais variáveis de ambiente, pode resultar em tempos de execuções diferentes, comprometendo a análise dos seus tempos de execução caso esta influência não tenha sido tratada. Note que essa alteração no tamanho de variáveis de ambiente pode ser realizada de forma intencional, pelo usuário, ou sem envolvimento prévio deste, por exemplo, sendo alterada por programas ou serviços do sistema operacional. Neste segundo caso, o não tratamento adequado da influência deste fator sobre os tempos de execução obtidos no experimento podem levar a erros de análise, levando a crer que diferenças observadas nos tempos de execução possam ser consequência de determinado fator sendo avaliado, enquanto de fato pode ser a influência de fatores externos intervenientes, tal como de variáveis de ambiente do programa sob teste.

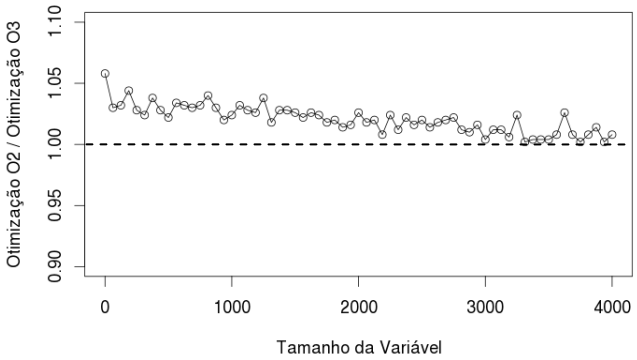

Fig. 4. Variabilidade do programa *CG* em relação ao tamanho da variável de ambiente.

C. Experimento #3

Ao analisar os resultados deste experimento observou-se que as amostras de todos os programas não seguem uma distribuição Gaussiana em pelo menos uma estratégia de *thread affinity*. Também, ao comparar os tratamentos para cada um dos programas, observou-se que há diferença significativa entre os tratamentos em pelo menos uma das comparações. Ou seja, o fator analisado tem influência sobre a variabilidade dos tempos de execução de alguns tratamentos. Como no Exp #1, não foi encontrado erro do tipo I nos resultados.

Ao analisar os tempos de execução dos programas, observou-se que permitir que o sistema operacional distribua as *threads* pelos núcleos (estratégia *no affinity*) mostra ser a melhor opção, pois em 90% dos casos o menor tempo de execução foi encontrado com esta estratégia. Para o maior tempo encontrado, observa-se que a estratégia *compact* se destacou em 96,67% dos casos. A Tabela V sumariza o resultado para cada estratégia de *thread affinity* avaliada.

TABELA V. DISTRIBUIÇÃO DO MAIOR E MENOR TEMPO POR ESTRATÉGIA DE THREAD AFFINITY

	No Affinity	**Compact**	**Scatter**
Maior Tempo	0,00%	96,67%	40,00%
Menor Tempo	90,00%	0,00%	3,33%

Também, constantou-se que houve variabilidade significativa nos tempos de execução de todos os programas do NPB, para um mesmo tratamento. Por limitação de espaço, a seguir é apresentado este resultado apenas do programa *MG*.

Para os tratamentos de 2 e 4 *threads*, observou-se uma pequena variabilidade nos tempos de execução do programa

MG. Contudo, para o tratamento com 6 *threads*, essa variabilidade se mostrou significativa para a estratégia *compact*. A Fig. 5 mostra a variabilidade dos tempos de execução do programa *MG* em relação às três estratégias de *thread affinity* para o tratamento com 6 *threads*. No caso do tratamento com a estratégia *compact* e 6 *threads*, sucessivas execuções (nas mesmas condições experimentais) produziram resultados diferentes de forma significativa, o que pode comprometer o resultado de estudos que não adotam múltiplas replicações do mesmo tratamento e/ou não adotam maior rigor estatístico ao analisar estes tempos de execução.

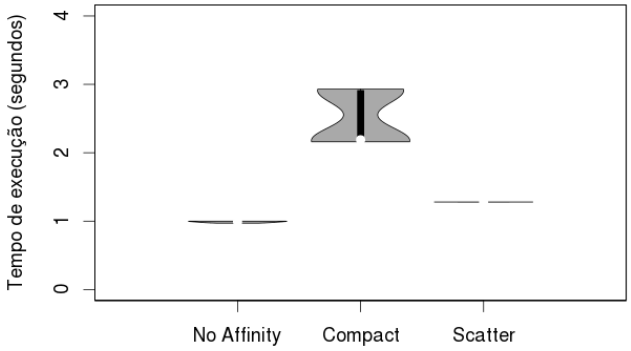

Fig. 5. Variabilidade dos tempos de execução do programa *MG*.

V. CONCLUSÕES

Experimentos computacionais são essenciais para a pesquisa científica. Contudo, a acuracidade dos seus resultados depende sobremaneira do rigor aplicado ao seu planejamento, execução e análise dos resultados. Neste trabalho, foram apresentados achados empíricos que deixam claro a importância em se considerar e tratar a variabilidade nos tempos de execução de programas de computador, os quais são influenciados por diferentes fatores ambientais, especialmente aqueles associados ao sistema operacional. Desconsiderar tais influências durante a análise dos resultados de tais experimentos implica em maior risco para a compreensão correta dos seus resultados.

Os achados aqui apresentados demonstram que o tamanho de uma variável de ambiente do sistema operacional pode influenciar, de forma significativa, os tempos de execução de programas. O mesmo foi observado para o nível de *Runlevel* e para a estratégia adotada para alocação de *threads*, muitas vezes não controlada em nível experimental e relegada ao compilador ou ao próprio SO. Importante salientar também que nem sempre as amostras de tempos de execução seguem uma distribuição Gaussiana, o que as vezes é adotado como presuposição por trabalhos na literatura.

Finalmente, foram propostas modificações ao método originalmente descrito em [1], para suportar a correta análise de tempos de execução provenientes de múltiplos tratamentos, de modo a identificar e corrigir o problema do aumento na probabilidade de erro tipo I para esses casos.

Como trabalhos futuros, é proposto o estudo de uma análise de significância para amostras provenientes de distribuições diferentes, bem como a análise de variabilidade inerente ao processo de coleta dos tempos de execução.

REFERÊNCIAS

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

[2] D. C. Montgomery, Design and Analysis of Experiments. John Wiley, 3rd ed., 2000.

[3] E. Vicente and R. Matias, "Exploratory Study on the Linux OS Jitter", In Proceedings of the 2012 Brazilian Symposium on Computing System Engineering, p. 19-24, Natal, 2012.

[4] A. Georges, D. Buytaert, and L. Eeckhout, "Statistically rigorous java performance evaluation," In Proceedings of the 22nd annual ACM SIGPLAN conference on Object-oriented programming systems and applications, p. 57–76, New York, 2007.

[5] T. Mytkowicz, A. Diwan, M. Hauswirth, and P. F. Sweeney, "Producing wrong data without doing anything obviously wrong!," In Proc. of the 14th international conference on Architectural support for programming languages and operating systems, p. 265–276, New York, 2009.

[6] A. Mazouz, S.-A.-A. Touati, and D. Barthou, "Study of variations of native program execution times on multi-core architectures," In Proc. of the 2010 International Conference on Complex, Intelligent and Software Intensive Systems, p. 919–924, Washington, 2010.

[7] A. Mazouz, S.-A.-A. Touati, and D. Barthou, "Analysing the variability of openmp programs performances on multicore architectures," In Fourth Workshop on Programmability Issues for Heterogeneous Multicores, p. 14, Heraklion, 2011.

[8] A. Mazouz, S.-A.-A. Touati, and D. Barthou, "Performance evaluation and analysis of thread pinning strategies on multi-core platforms: Case study of SPEC OMP applications on intel architectures," High Performance Computing and Simulation. p. 273-279 , Istanbul, 2011.

[9] K. K. Pusukuri, R. Gupta, and L. N. Bhuyan,. "Thread tranquilizer: Dynamically reducing performance variation," ACM Trans. Archit. Code Optim., 8, 4, January 2012, 46:1-46:21.

[10] D. J. Lilja, Measuring Computer Performance: A Practitioner's Guide. Cambridge University Press, 2005.

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

[12] M. Kerrisk, "Linux User's Manual", http://man7.org/linux/man-pages/man1/time.1.html, 2014.

[13] NASA Advanced Supercomputing Divison, "NAS Parallel Benchmarks", http://www.nas.nasa.gov/publications/npb.html, 2014.

[14] GCC, "GCC, the GNU Compiler Collection", https://gcc.gnu.org/, 2014.

[15] ICC, "Intel® C and C++ Compilers", https://software.intel.com/en-us/c-compilers/, 2014.

[16] GNU, "GNU libomp", https://gcc.gnu.org/onlinedocs/libgomp/, 2014.

[17] P. Barbetta, M. Reis, and A. Bornia, Estatística: para cursos de engenharia e informática. Atlas S.A., 3ª ed., 2010.

[18] D. C. Howell, Statistical Methods for Psychology. Cengage Learning, 7th ed., 2009.

[19] S. Glantz, Primer of Biostatistics, Seventh Edition. McGraw-Huill Education, 7th ed., 2014.

[20] S. S. Shapiro, and M. B. Wilk, "An analysis of variance test for normality (complete samples)". Biometrika, December 1965, 52(3/4):591–611.

[21] J. D. Gibbons and S. Chakraborti. Nonparametric Statistical Inference Fourth Edition, Revised and Expanded. Marcel Dekker, Inc., 4th ed., 2003.