UNIVERSITE DE VERSAILLES SAINT-QUENTIN EN YVELINES

The Speedup Test

Sid-Ahmed-Ali TOUATI — Julien WORMS — Sebastien BRIAIS

N° HAL-inria-00443839

January 2010

Rapport technique

The Speedup Test

Sid-Ahmed-Ali Touati* , Julien Worms † , Sebastien Briais ‡

Thème : Optimisation de programmes
Équipe-Projet ALCHEMY. PROBA. ARPA.

Rapport technique n° HAL-inria-00443839 — January 2010 — 35 pages

Abstract: Numerous code optimisation methods are usually experimented by doing multiple observations of the initial and the optimised executions times in order to declare a speedup. Even with fixed input and execution environment, programs executions times vary in general. So hence different kinds of speedups may be reported: the speedup of the average execution time, the speedup of the minimal execution time, the speedup of the median, etc. Many published speedups in the literature are observations of a set of experiments. In order to improve the reproducibility of the experimental results, this technical report presents a rigorous statistical methodology regarding program performance analysis. We rely on well known statistical tests (Shapiro-wilk's test, Fisher's F-test, Student's t-test, Kolmogorov-Smirnov's test, Wilcoxon-Mann-Whitney's test) to study if the observed speedups are statistically significant or not. By fixing $0 < \alpha < 1$ a desired risk level, we are able to analyse the statistical significance of the average execution time as well as the median. We can also check if $\mathbb{P}[X > Y] > \frac{1}{2}$, the probability that an individual execution of the optimised code is faster than the individual execution of the initial code. Our methodology defines a consistent improvement compared to the usual performance analysis method in high performance computing as in [15, 11]. We explain in each situation what are the hypothesis that must be checked to declare a correct risk level for the statistics. The Speedup-Test protocol certifying the observed speedups with rigorous statistics is implemented and distributed as an open source tool based on `R` software.

Key-words: Code optimisation, program performance evaluation and analysis, statistics

* INRIA-Saclay. Sid.Touati@inria.fr
† Laboratoire de mathématiques de Versailles. Julien.Worms@math.uvsq.fr
‡ Laboratoire PRiSM. Sebastien.Briais@prism.uvsq.fr

UFR des sciences
45 avenue des Etats Unis,
78035 Versailles cedex
Téléphone : +33 1 39 25 41 12 — Télécopie : +33 1 39 25 46 04

Un protocole statistique pour l'analyse de l'accélération des programmes

Résumé : De nombreuses techniqus d'optimisations de programmes sont expérimentées en mesurant plusieurs fois les temps d'exécutions du code initial et du code transformé. Même en fixant les données d'entrés et l'environment d'exécution, les temps observés pour les exécutions des programmes sont variables en général. Ainsi, plusieurs facteurs d'accélérations possibles peuvent être observés: accélération du temps minimum, accélération du temps moyen et accélération du temps médian. Ces observations ne sont pas toujours significatives statistiquement. Afin d'améliorer la reproducibilité des performances des programmes, nous présentons dans ce document une méthodologie statistique rigoureuse basée sur plusieurs tests connus (test de Shapiro-wilk, test F de Fisher, test de Student, test de Kolmogorov-Smirnov, test de Wilcoxon-Mann-Whitney's). En se fixant un niveau de risque α souhaité, nous somme capabls de comparer entre deux moyennes ou deux médianes variables. Notre méthodologie définit une amélioration par rapport aux protocols usuels décrits dans [11, 15]. Par ailleurs, nous expliquons dans chaque situation d'observation d'accélération quelles sont les hypothèses à vérifier pour déclarer un niveau de risque correct. Le protocole statistique, appelé le *Speedup-Test*, certifiant que les accélérations observées sont statistiquement valides est distribué sous forme de logiciel libre basé sur `R`.

Mots-clés : Optimisation de code, analyse et évaluation des performances des programmes, statistiques

Contents

1 Introduction

The community of program optimisation and analysis, code performance evaluation, parallelisation and optimising compilation has published since many decades numerous research and engineering articles in major conferences and journals. These articles study efficient algorithms, strategies and techniques to accelerate programs execution times, or optimise other performance metrics (MIPS, code size, energy/power, MFLOPS, etc.). The efficiency of a code optimisation technique is generally published according to two principles, not necessarily disjoint. The first principle is to provide a mathematical proof given a theoretical model that the published research result is correct or/and efficient: this is the hard part of research in computer science, since if the model is too simple, it would not represent real world, and if the model is too close to real world, mathematics become too complex to digest. A second principle is to propose and implement a code optimisation technique and to practice it on a set of chosen benchmarks in order to evaluate its efficiency. This article concerns this last point: how can we convince the community by rigorous statistics that the experimental study publishes fair and reproducible results.

1.1 Known hints for making a research result non reproducible

Hard natural sciences such as physics, chemistry and biology impose strict experimental methodologies and rigorous statistical measures in order to guarantee the reproducibility of the results with a measured confidence (probability of error/success). The reproducibility of the experimental results in our community of program optimisation is a weak point. Given a research article, it is in practice impossible or too difficult to reproduce the published performance. If the results are not reproducible, the benefit of publishing becomes limited. We note below some hints that make a research article non-reproducible:

- Non using of precise scientific languages such as mathematics. Ideally, mathematics must always be preferred to describe ideas, if possible, with an accessible difficulty.
- Non available software, non released software, non communicated precise data.
- Not providing formal algorithms or protocols make impossible to reproduce exactly the ideas. For instance, the authors in [3] spent large efforts and time to re-implement some hardware branch predictor mechanisms based on previous published research articles, but they fail to reproduce them exactly. Simply because the initial articles describing the branch predictors are not formal, so they can be interpreted and implemented differently by external readers.
- Hide many experimental details. As demonstrated by [17], bringing small modification on the execution environment brings contradictory experimental results. For instance, just changing the size of the linux shell variables or the order of linking an application alter the conclusions. As pointed by the authors in [17], a lot of published articles in major conferences hide these details.
- Usage of deprecated machines, deprecated OS, exotic environment, etc. If we take a research article published five years after the experiments for instance, there is a high chance that the workstations that served the experiments have already died or already changed their behaviour (usury of hardware, software patches, etc.).
- Doing wrong statistics with the collected data.

Part of the non-reproducibility (and not all) of the published experiments is explained by the fact that the observed speedups are sometimes *rare* events. It means that they are far from what we could observe if we redo the experiments multiple times. Even if we take an ideal situation where we use exactly the original experimental machines and software, it is sometimes difficult to reproduce exactly the same performance numbers again and again, experience after experience. Since some published performances numbers represent exceptional events, we believe that if a computer scientist succeeds in reproducing the performance numbers of his colleagues (with a reasonable error ratio), it would be equivalent to what rigorous probabilists and statisticians call a *surprise*. We argue that it is better to have a lower speedup that can be reproduced in practice, than a rare speedup that can be remarked by accident.

1.2 Why program execution times vary

What makes a binary program execution time to vary, even if we use the same data input, the same binary, the same execution environment? Here are some factors: background tasks, concurrent jobs, OS process scheduling, binding (placement) of threads on cores/processors, interrupts, input/output, starting loader address, starting execution stack address, branch predictor initial state, cache effects, non deterministic dynamic instruction scheduler, temperature of the room (dynamic voltage/frequency scaling service), bias or imprecision in performance measurement tools, etc.

One of the reasons of the non-reproducibility of the results, and not only, is the variation of execution times of the same program given the same input and the same experimental environment. With the massive introduction of multicore architectures, we observe that the variations of executions times become exacerbated because of the complex dynamic features influencing the execution [12]: threads scheduling policy, synchronisation barriers, resource sharing between threads, hardware mechanisms for speculative execution, etc. Consequently, if you execute a program (with a fixed input and environment) n times, it is possible to obtain n really distinct execution times [12]. The mistake is to always assume that these variations are minor, and are stable in general. The variation of execution times is something that we observe everyday, we cannot neglect it, but we can analyse statistically with rigorous methodologies. An usual error in the community is to replace all the n execution times by a single value, such that the minimum, the mean or the maximum, losing any data on the variability. Note that reporting the variance of a sample of program executions is helpful but not sufficient, because it does not allow to measure the chance of observing the same variance in future samples of executions.

1.3 Related literature on statistical performance analysis

Program performance analysis and optimisation may rely on two well known books that explain digest statistics to our community [15, 11] in an accessible way. These two books are good introductions for doing fair statistics for analysing data. Based on these two books, previous work on statistical program performance evaluation have been published [5, 16].

However, [5, 11, 15, 16] focus on the average execution times only. Since the average is known to not be a good performance measure (since the average is sensitive to outliers), the median is usually advised for reporting performance numbers (such as for SPEC scores). Consequently, we rely on more academic books on statistics [6, 8, 13] for comparing between two medians. Furthermore, these fundamental books help us to understand mathematically some common mistakes and misunderstanding of statistics.

1.4 What is inside this document, what are its limitations

We base our reasoning here on common well known results in mathematical statistics [6, 8, 13], as well as on known books in practice of performance analysis [15, 11]. This documents presents a rigorous statistical methodology to evaluate program optimisation techniques. This document recalls some common mistakes in statistical performance evaluation, explains which statistics should be used under which hypothesis, and provides practical examples. Our examples are based on the free software called `R` [2, 14]: it is a complete and free software for statistics with an easy-to-learn high level language.

Our document is organised to help computer scientists willing to make correct and rigorous statistical study of their code optimisation method. The question is how to convince real experts by correct statistics, provided a risk level $\alpha \in]0\%, 100\%[$ (conversely a confidence level $1 - \alpha$), that a code optimisation technique is really efficient in practice, and the speedup it produces for a given benchmark is statistically significant.

Our technical document is organised as follows. Section 2 illustrates some usual miss-use or lack or rigour in performance analysis (we will not cite any article in this situation). Section 3 recalls the exact definitions of different kinds of speedups, as well as overall speedups of a set of benchmarks. Section 4 shows two protocols to decide, with a risk level $0 < \alpha < 1$, if a speedup is statistically significant or not: we study the two cases of the speedups of the average and the median execution time. Getting a speedup (acceleration) inside a sample of b benchmarks does not guarantee us that we can get a speedup on another program outside the selected set of benchmarks. Consequently, Section 5 shows how we can estimate the chance that the code optimisation would provide a speedup on a program not belonging to the initial sample of benchmarks used for experiments. All the statistical methods presented in this

document has been implemented in a free software available for the community, called the `SpeedupTest`: Section 6 gives the technical manual of the software.

In this document, we have put some important discussions and analysis in the appendix part. Appendix A explains why considering the minimal observed execution time is not a fair performance metric for a program. Appendix B explains the notion of hypothesis testing in statistics with a risk level α. We also explain why the common sense of the confidence level $1-\alpha$ is not exactly the correct mathematical one. Appendix C is a discussion and a study about the question: *How large should be a sample size ? How many runs do we need ?*.

The limitations of this document are: we do not study the variations of the executions times due to changing the program's input. We consider observations of real executions, not emulation or simulation. We also consider that the observations are done on a fixed (universal ?) experimental environment.

1.5 Remark on the collected data (observations of executions times)

The reported executions times are considered with a *continuous* time unit (for instance in seconds and its fractions) not with discrete time units (time steps, clock cycles, etc.). If the execution time has been observed in clock cycles (by hardware performance counters for instance), the statistical procedures assume that the reported values are continuous. The reason is that the statistical models we use in this document assume continuous random variables and not discrete ones. If the data we analyse are assumed discrete, the statistics described in this document become invalid.

Note that transforming the observed clock cycles to the continuous time (by dividing the observed number of the clock cycles by the CPU frequency) would only change the scale of the data. This transformation does not change the nature of the data. All the statistics described in this document are valid for discrete obervations (clock cycles, time steps, etc.) if the user accepts that the time unit is continuous and not discrete.

1.6 The notations used in the document

Let $\mathcal{C}$ be an initial code, let $\mathcal{C}'$ be a transformed version after applying the program optimisation technique under study. Let I be a fixed input. If we execute the program $\mathcal{C}(I)$ n times, it is possible that we obtain n distinct executions times (especially if the code is a kernel, or a toy benchmark). Let X be the random variable representing the execution time of $\mathcal{C}(I)$. Let $\mathcal{X} = \{x_1, \cdots, x_n\}$ be a sample of observations of X, *i.e* the set of the observed executions times when executing $\mathcal{C}(I)$ n times. The transformed code $\mathcal{C}'$ can be executed m times with the *same* data input I producing m execution times too. Similarly, let Y be the random variable representing its execution time. Let $\mathcal{Y} = \{y_1, \cdots, y_m\}$ be a sample of obervations of Y, it contains the m observed execution times of the code $\mathcal{C}'(I)$.

The theoretical mean of X and Y are noted μ_X and μ_Y respectively. The theoretical medians of X and Y are noted $\text{med}(X)$ and $\text{med}(Y)$. The theoretical variances are noted σ_X^2 and σ_Y^2. The cumulative distribution functions (CDF) are noted $F_X(a) = \mathbb{P}[X \leq a]$ and $F_Y(a) = \mathbb{P}[Y \leq a]$, where $\mathbb{P}[X \leq a]$ is the notation used for the probability that $X < a$. The probability density functions are noted $f_X(x) = \frac{dF_X(x)}{dx}$ and $f_Y(y) = \frac{dF_Y(y)}{dy}$.

Since X and Y are two samples, their sample mean are noted $\bar{X} = \frac{\sum_i x_i}{n}$ and $\bar{Y} = \frac{\sum_j y_j}{m}$ respectively. The sample variances are noted s_X^2 and s_Y^2. The sample medians of X and Y are noted $\overline{\text{med}}(X)$ and $\overline{\text{med}}(Y)$. The sample median is defined as follows; The observations x_i are sorted in ascending order $X = \{x_{(1)}, \cdots, x_{(n)}\}$, where $x_{(k)}$ is the k^{th} sorted value of X (x_i and $x_{(i)}$ are two distinct values of the same sample). Then $\overline{\text{med}}(X) = x_{(\lceil n/2 \rceil)}$ if n is odd, otherwise $\overline{\text{med}}(X) = \frac{x_{(n/2)} + x_{(1+n/2)}}{2}$.

2 Common observed non rigorous experiments and statistics in the community of code optimisation and high performance computing

This section shows some observed non rigorous statistical study in the litterature of our community. We are not willing to reference any article in this situation. We admit that statistics theory is a difficult discipline, and we are all good candidate to make mistakes. This section is willing to highlight why many published speedups cannot be reproduced easily.

2.1 Biased experiments

Contrary to natural science, the community of high performance computing did not decide for a common protocole for measuring program performance. While SPEC organism provides some advices to do performance evaluation, following them is not currently an obligation for publication. We can report here some situations known to produce biased experiments:

- It is important that the repetitive observed executions of the same program must be *independent*. For instance, it is not fair to use a single loop around a code kernel that repeats the execution k times. This is because repeating a code kernel $\mathcal{C}$ inside a loop of k iterations makes it to execute inside the same application. Consequently, the operating system does not behave as if you execute the program k times from the shell. Furthermore, the caches are warmed by the repetitive executions of the code kernels if they belong to the same application.

- Even if we execute a program k times from the shell, the executions are not necessarily independent, especially if they are executed back-to-back: the seek time of the disk is altered by repetitive executions, some data are cached on the disk by applications and benefit from repetitive executions. Recently, we have been told that branch predictors are also influenced by separate applications: this seems strange, but we should stay careful with micro-architectural mechanisms.

- Long running time applications increase the temperature of the machine. Consequently, some OS services, such as dynamic voltage scaling (DVS), are activated to reduce the temperature by reducing the CPU frequency. If the CPU frequency is reduced, the execution time is altered.

- If the program under study (a code fraction, function, kernel, etc.) is executed with multiple distinct inputs, each input should produce its own set of performance data. That is the pairs (C_k, I_1) (C_k, I_2) should be made distinct. It is not fair to compute an average execution time among all the inputs, because such average would smooth the performance data among all inputs, consequently the variance is smoothed too, which yields to a loss in statistical information[1]

As can be remarked from the previous points, it is not easy to design a *ideal* experimental environment. Many hidden factors influencing the executions times are not well known yet. For instance, just changing the size of the linux shell variables or the order of linking an application alter the conclusions. As pointed by the authors in [17], a lot of published articles in major conferences hide these details. Ideally, we must guarantee that the observed execution times are completely independent. For instance, rebooting the entiere system before each individual run would guarantee such independance, but would increase the cost of the experiments. What we can advice however is to deliver detailed information about the protocole used for experiments: overhead of the system (low overhead, multiple users, etc.), activated services (dynamic voltage scaling must be desactivated), the exact way of executions (from the shell or not, variable size environment, back-to-back runs, reboot or not, etc.), the used OS and the compiler flags, etc. With all these details, a careful expert will have enough information to reproduce the experiments.

[1] The variability of execution times when the data input varies cannot be analysed with probability theory easily. Simply because when data input varies, the execution time varies inherently based on the algorithmic complexity, and not because of the structural hazard. In other words, observing distinct execution times when varying data input cannot be considered as hazard, but as an inherent reaction of the program under analysis.

2.2 Outliers elimination

In natural science, when an observation fails (a dead mouse, a failure in measurement), researchers remove the outliers from database. In program performance analysis, we *strongly advise to keep all the observed executions times*, unless the program crashes or produces wrong results (in this situation we can remove the outliers corresponding to crashes or bugs). Keeping all the performance data of correct executions is important for statistical analysis since it brings more confidence and more information. Our arguments against removing outliers are as follows:

1. Nothing guarantee that the outlier (min or max) is an accident of the experimental activity.
2. If an outlier is a rare event, it shouldn't appear in the sample;
3. If an outlier appears in the sample, then it is not a rare event;
4. We know how to make fair statistics that are less sensitive to outliers.

2.3 Using statistical tests without checking the hypothesis

The most common observed mistake is to use the Student's t-test without checking the normality of distribution for small samples. Indeed, the Student's t-test is proved only for gaussian distribution [6, 13]. If the data are not normal, the Student's t-test computes a wrong risk level. However, it is admitted (thanks to the central limit theorem) that the test remains robust for large non gaussian samples, but the computed risk level is not preserved (page 71 of [8]): if the sample is large enough, the computed risk level should not be too far from the correct one[2]. Computing the correct risk level for a non gaussian distributions can be done if the distribution of the data is known.

2.4 Confusion between continuous and discrete data

When doing performance analysis, it is important to distinguish between two general cases: continuous data and discrete data. If we analyse execution times with a continous time unit, then we can consider the continuous model. All the statistical tests that will presented in this document are proved in the continuous model. If the data are assumed discrete (for instance, the number of clock cycles, number of registers, counted categories, etc), we cannot use correctly the statistical tests presented in this document. Consequently, if the executions times have been observed by hardware performance counters reporting clock cycles, we must assume them as observations of continous time (since the execution time in seconds is simply the observed number of clock cycles divided by the CPU frequency). For instance, if we observe that the execution time in clock cycles is the sample $\{1, 3, 3, 5, 2, 4, 4\}$, then the statistical tests presented in this document assume that this sample is from a continuous distribution $\{1.0, 3.0, 3.0, 5.0, 2.0, 4.0, 4.0\}$, so all possible real values are considered to be observables: this is not true in practice because hardware performance counters observe discrete events in general.

[2]Thanks to the central limit theorem, the quantity $\left|\mathbb{P}\left[\sqrt{n}\frac{\bar{X}-\mu_X}{s_X.n} \leq x\right] - \mathbb{P}\left[t_{(n-1)} \leq x\right]\right|$ remains bounded for any continuous distribution function followed by X (assuming a finite theoretical variance). The bound of the error depends on the distribution function of X.

3 Reporting the different kinds of observed speedups

3.1 Computing a speedup for a single program with a single data input

After measuring X and Y the executions times of the codes $\mathcal{C}(I)$ and $\mathcal{C}'(I)$ respectively for the same data input I, a simple definition of the speedup [9] sets it as $\frac{X}{Y}$. In reality, since X and Y are random variables, the definition of a speedup becomes more complex. Ideally, we must analyse the probability density functions of X, Y and $\frac{X}{Y}$ to decide for a speedup or not. Since this is not an easy problem, multiple sorts of observed speedups are usually reported in practice to simplify the performance analysis:

1. The observed speedup of the minimal execution times:

$$\texttt{spmin}(\mathcal{C}, I) = \frac{\min_i x_i}{\min_j y_j}$$

2. The observed speedup of the mean (average) execution times:

$$\texttt{spmean}(\mathcal{C}, I) = \frac{\bar{X}}{\bar{Y}} = \frac{\sum_{1 \leq i \leq n} x_i}{\sum_{1 \leq j \leq m} y_j} \times \frac{m}{n}$$

3. The observed speedup of the median execution times:

$$\texttt{spmedian}(\mathcal{C}, I) = \frac{\overline{\text{med}}(X)}{\overline{\text{med}}(Y)}$$

In the literature of high performance computing, it is not always clear which one of the above speedups is reported. Usually, the community publishes the best speedup among those observed, without any guarantee of reproducibility. Below our opinions on each of the above speedups:

- Regarding the observed speedup of the minimal execution times, we do not advise to use it for many reasons. Appendix A explains why using the observed minimal execution time is not a fair choice regarding the chance of reproducing the result.
- Regarding the observed speedup of the mean execution time, it is well understood in statistical analysis but remains sensitive to outliers. Consequently, if the program under optimisation study is executed few times by an external user, the latter may not able to observe the reported average.
- Regarding the observed speedup of the median execution times, it is the one that is used by the SPEC organisation. Indeed, the median is a better choice for reporting speedups, because the median is less sensitive to outliers. Furthermore, most of the practical cases show that the distribution of the executions times are skewed, making the median a better candidate for summarising the executions times into a single number.

All the above speedups are observation metrics, that do not guarantee their reproducibility. Another definition of a speedup is to test whether $X > Y$, neither in average nor by its median, but by considering if an individual run x_i is higher or not than an individual run y_j. Later, we will explain a statistical test that confirms or not whether $\mathbb{P}[X > Y] > \frac{1}{2}$, *i.e.* the chance that $x_i > y_j$ is greater than $\frac{1}{2}$.

3.2 Reporting the overall speedup and performance gain of a set of benchmarks

When we implement a code optimisation technique, we are generally asked to test it on a set of benchmarks, not on a unique one. Let b be the number of considered benchmarks. Ideally, the code optimisation technique should produce speedups on the b programs (at least no slowdown). Unfortunately, this situation is rare nowadays. Usually, only a fraction of a programs among b would benefit from an acceleration. Let $\texttt{speedup}(\mathcal{C}_k, I_k)$ $(1 \leq k \leq b)$ be the obtained speedup for the k^{th} code $\mathcal{C}_k$ (see Section 3.1 for the different kinds of observable speedups). While it is not always justified, we can be asked to measure an *overall* speedup of all the benchmarks. In statistics, we cannot provide a fair average because the

programs are different, and their weights are different too. So, asking for an overall speedup for a set of benchmarks will highly bring an unfair value. Neither an arithmetic mean, nor a geometric or harmonic mean can be used to synthesise in a unique speedup of the whole set of benchmarks.

The arithmetic mean of the observed speedups does not distinguish between short and long programs: for instance, having a speedup of 105% on a program which lasts 3 days must not have the same impact as a speedup of 300% obtained on a program which lasts 3 seconds. In the former, we save 5% of 3 days (=216 minutes), while in the latter we save 200% of 3 seconds (=2 seconds). If we use the arithmetic mean, we would obtain an overall speedup equal to (105+300)/2=202%, this does not reflect the reality with a fair number.

The geometric mean cannot be applied here because we are not faced to a succession of accelerations on the same program, but we are faced to accelerations of distinct programs. The harmonic mean is not meaningful too because the quantity $\frac{1}{\texttt{speedup}(\mathcal{C}_k, I_K)}$ also represents a sort of speedup, so we can provide the same criticism as for the arithmetic mean .

However, we can still compute S a sort of overall speedup, as well as we can compute G an overall *performance gain factor* (not an overall speedup). Both the S and G can consider the weights of the different programs. We can use the following method.

First, an interesting question is to decide whether we should neglect the $b-a$ programs with slowdown. That is, S and G are computed for only the subset a of the benchmarks, not on all the b benchmarks. We believe that we can neglect the $b-a$ programs with slowdown if we study afterwards (in Section 5) the confidence interval of the proportion $\frac{\mu_a}{b}$. This fractions represents the probability that the code optimisation produces a speedup. sStudying this proportion helps us to decide if the reported overall gain is meaningful. If we decide to include all the b programs for computing S and G, this is also fair, but the reported G may be negative since it includes the slowdowns, and the reported S may be < 1.

Second, we associate a weight $W(\mathcal{C}_k)$ to each program $\mathcal{C}_k$. The general characteristics of a weight function is $\sum_{1\leq k\leq b} W(\mathcal{C}_k) = 1$. If not, we should normalise the weights so that they sum to 1. The weight of each benchmark can be chosen by the community, by the benchmark organisation, by the user, or we can simply decide to associate the same weight to all benchmarks. Also, we can also choose the weight as the fraction between the observed execution time and the sum of all observed execution times.

$$W(\mathcal{C}_k) = \frac{\texttt{ExecutionTime}(\mathcal{C}_k, I_k)}{\sum_{i=1,b} \texttt{ExecutionTime}(\mathcal{C}_i, I_i)} \tag{1}$$

where $\texttt{ExecutionTime}(\mathcal{C}_i, I_i)$ is the considered execution time of the code C_i having I_i as data input. We choose to compute $\texttt{ExecutionTime}(\mathcal{C}_i, , I_i)$ with one of the usual functions (mean, median, min) *i.e.*, the mean or the median or the min of the observed execution times of the code $\mathcal{C}_i$. Someone would argue that this would give more weight on long running time programs: the answer is yes, and this has a sense if want to optimise the absolute execution time, not the relative one. Anyway, choosing a weight for a given application is a matter of discussion, and every user is free to fix it according to its own situation.

Third, transforming a program $\mathcal{C}_k$ into $\mathcal{C}'_k$ allows to reduce the execution time by $\texttt{ExecutionTime}(\mathcal{C}_k, I_k) - \texttt{ExecutionTime}(\mathcal{C}'_k, I_k)$. This absolute gain should not be considered as it is, but should be multiplied by the weight of the program as follows: $g(\mathcal{C}_k) = W(\mathcal{C}_k) \times (\texttt{ExecutionTime}(\mathcal{C}_k, I_k) - \texttt{ExecutionTime}(\mathcal{C}'_k, I_k))$.

Fourth and last, the overall performance gain factor is defined as the fraction between weighted gains and the sum of weighted initial execution times: $G = \frac{\sum_{j=1,b} g(\mathcal{C}_j)}{\sum_{j=1,b} W(\mathcal{C}_j) \times \texttt{ExecutionTime}(\mathcal{C}_j, I_j)}$. By simplification, we obtain:

$$G = 1 - \frac{\sum_{j=1,b} W(\mathcal{C}_j) \times \texttt{ExecutionTime}(\mathcal{C}'_j, I_j)}{\sum_{j=1,b} W(\mathcal{C}_j) \times \texttt{ExecutionTime}(\mathcal{C}_j, I_j)} \tag{2}$$

By definition, the overall gain $G < 1$, since the execution times of the optimised programs are positive values ($\texttt{ExecutionTime}(\mathcal{C}'_j, I_j) > 0$). The overall speedup can be computed as follows:

$$S = \frac{\sum_{j=1,b} W(\mathcal{C}_j) \times \texttt{ExecutionTime}(\mathcal{C}_j, I_j)}{\sum_{j=1,b} W(\mathcal{C}_j) \times \texttt{ExecutionTime}(\mathcal{C}'_j, I_j)} \tag{3}$$

Example 3.1. *Let a program P1 that initially lasts with a median of 3 seconds. Assume we succeed to accelerate it with a factor of 300%. Thus, its new median execution time becomes 1 second. Let P2 be a program that initially lasts 1 hour and has been accelerated with a factor of 105%. Thus, its new median*

execution time becomes 3428 seconds. The arithmetic mean of these two speedups is 202.5%, the geometric mean is 177.48% and the harmonic mean is 155.56%. None of these means is suggested for performance summary for the reasons explained before. We choose to fix the weights of the programs P1 and P2 as $W(P1) = 3/(3600 + 3) = 0.0008$ *and* $W(P2) = 3600/(3600 + 3) = 0.9991$*. Other weights are possible, depending on the situation of each user. The obtained weighted gain for each program is:* $g(P1) = 0.001$ *and* $g(P2) = 171.85$*. The overall performance gain factor is then* $G = 1 - \frac{0.0008 \times 1 + 0.9991 \times 3428}{0.0008 \times 3 + 0.9991 \times 3600} = 4.77\%$*. If we consider that the weights are equal,* $W(P1) = W(P2) = 1$*, then the overall performance gain factor is then* $G = 1 - \frac{1+3428}{3+3600} = 4.82\%$ *and* $S = \frac{3+3600}{1.3426} = 1.05$*. As can be remarked, there is not a direct comparison between the overall gain and the individual speedups.*

4 Analysing the statistical significance of the observed speedups

The observed speedups are performance numbers observed once (or multiple times) on a sample of executions. Does this mean that the future executions would conclude with speedups ? How can we be sure about this question if no mathematical proof exists, and with which confidence level ? This section answers these questions. For the rest of this section, we define $0 < \alpha < 1$ as the risk (probability) of error (making a wrong conclusion). Conversely, $(1 - \alpha)$ is the usual confidence level. Usually, α is a small value (for instance $\alpha = 5\%$).

The user must be aware that in statistics, the risk of error is included in the model, so we are not always able to decide between two contradictory situations (as in logic where we can decide between true and false). Furthermore, the abuse of language defines $(1 - \alpha)$ as a confidence level, while this is not exactly true in the mathematical sense. Indeed, there are two types of risks when we use statistical tests, see Appendix B. Often, we say that a statistical test (normality test, Student's test, etc.) concludes favourably by a confidence level $(1 - \alpha)$ because it didn't succeed to reject the tested hypothesis with a risk level equal to α. When a statistical test does not reject an hypothesis with a risk equal to α, there is usually no proof that the contrary is true with a confidence level of $(1 - \alpha)$. This way of reasoning is admitted for all statistical tests since in practice it works well. Appendix B gives more details on hypothesis testing in statistics.

4.1 The speedup of the observed average execution time

Having two samples X and Y, deciding if μ_X the theoretical mean of X is higher than μ_Y the theoretical mean of Y with a confidence level $1-\alpha$ can be done thanks to the Student's t-test [15]. In our situation, we use the one-sided version of the Student's t-test and not the two sided version (since we want to check whether the mean of X is higher than the mean of Y, not to test if they are simply distinct). Furthermore, the observation x_i does not correspond to another observation y_j, so we use the unpaired version of the Student's t-test.

Remark on the normality of the distributions of X and Y The mathematical proof of the test of Student is valid for Gaussian distributions only [6, 13]. If X and Y are not from Gaussian distributions (normal is synonymous to Gaussian), then the test of Student is know to stay robust for large samples (thanks to the central limit theorem), but the computed risk α is not exact [13, 6]. If X and Y are not normally distributed and are small samples, then we cannot conclude with the Student's t-test.

Remark on the variances of the distributions of X and Y In addition to the Gaussian nature of X and Y, the original Student's t-test was proved for populations with the same variance ($\sigma_X^2 \approx \sigma_Y^2$). Consequently, we also need to check whether the two populations X and Y have the same variance by using the Fisher's F-test for instance. If the Fisher's F-test concludes that $\sigma_X^2 \neq \sigma_Y^2$, then we must use a variant of Student's t-test that considers Welch's approximation of the degree of freedom.

The size needed for the samples X and Y The question now is to know what is a *large* sample. Indeed, this question is complex and cannot be answered easily. In [11, 15], a sample is said large when its size exceeds 30. However, that size is well known to be arbitrary, it is commonly used for a numerical simplification of the test of Student[3]. Note that $n > 30$ is not a size limit needed to guarantee the robustness of the Student's t-test when the distribution of the population is not Gaussian, since the t-test remains sensitive to outliers in the sample. Appendix C gives a discussion on the notion of *large* sample. In order to set the ideas, let us consider that $n > 30$ defines the size of large samples. Appendix C shows that this assumption is reasonable in practice.

[3]When $n > 30$, the Student distribution begins to be correctly approximated by the standard Gaussian distribution, allowing to consider z values instead of t values. This simplification is out of date, it has been made in the past when statistics used to use pre-computed printed tables. Nowadays, computers are used to numerically compute real values of all distributions, so we do no longer need to simplify the test of Student for $n > 30$. For instance, the current implementation of the Student's t-test in the statistical software `R` does not distinguish between small and large samples, contrary to what is explained in [15, 11].

Using the Student's t-test correctly H_0, the null hypothesis that must be rejected by the Student's t-test is that $\mu_X \leq \mu_Y$, with an error probability equal to α. If the test rejects this null hypothesis, then we can accept H_a the alternative hypothesis $\mu_X > \mu_Y$ with a confidence level $1-\alpha$ (Appendix B explains the exact meaning of the term *confidence level* used in this document). The Student's t-test computes a p-value, which is the smallest probability of error to reject the null hypothesis. If p-value$\leq \alpha$, then the Student's t-test rejects H_0 with a risk level lower than α. Hence we can accept H_a with a confidence level $(1-\alpha)$ [4].

As explained before, using correctly the Student's t-test is conditioned by:

1. If the two samples are large enough (say $n > 30$ and $m > 30$), using the Student's t-test us admitted but the computed risk level α may still be inaccurate if the underlying distributions of X and Y are too far from being normally distributed not preserved (page 71 of[8]).

2. If one of samples is small (say $n \leq 30$ and $m \leq 30$)

   (a) If X or Y does not follow Gaussian distributions with a risk level α, then we cannot conclude about the statistical significance of the observed speedup of the average execution time.

   (b) If X and Y follow Gaussian distributions with a risk level α than:

   - If X and Y have the same variance with a risk level α then use the original procedure of the test of Student.
   - If X and Y do not have the same variance with a risk level α then use the Welch's version of the Student's t-test procedure.

The detailed description of the Speedup-Test protocol for the average execution time is illustrated in Figure 4.1.

Example (done with R) 1. *Let $\mathcal{C}$ be a initial program with its representative data input. We are willing to statistically demonstrate with a risk level $\alpha = 5\%$ that an optimisation technique transforms it into $\mathcal{C}'$ and produces benefit in terms of average execution speed. For doing this, we should execute $\mathcal{C}$ and $\mathcal{C}'$ a large number of times (more than 30 times) with the same data input. For the sake of the example, we consider here only 5 executions for $\mathcal{C}$ and $\mathcal{C}'$. By using the R software [14], we introduce the values of execution times (in continuous time unit) of $\mathcal{C}$ and $\mathcal{C}'$ as two sample vectors $\mathcal{X}$ and $\mathcal{Y}$ respectively*

```
> X<- c(2.799, 2.046, 1.259, 1.877, 2.244)
> Y <- c(1.046, 0.259, 0.877, 1.244, 1.799)
```

In the following, we will statistically check based on these two samples that $\mu_X > \mu_Y$ with a confidence level $1-\alpha = 95\%$. Since we have only 5 observations instead of more than 30, we must check the normality of the values of X and Y. By using the test of Shapiro-Wilk:

```
> shapiro.test(X)
Shapiro-Wilk normality test
data:  T1
W = 0.9862, p-value = 0.9647
```

The test of Shapiro-Wilk on the data $\mathcal{X}$ computes p-value= 0.9647. This value is the lowest risk probability to reject the normality. Since p-value> α, we do not reject the normality assumption, so we can consider that X follows a Gaussian distribution. By performing the same test on $\mathcal{Y}$:

```
> shapiro.test(Y)
Shapiro-Wilk normality test
data:  T2
W = 0.9862, p-value = 0.9647
```

[4] Appendix B gives more details on hypothesis testing in statistics, and on the exact meaning of the confidence level.

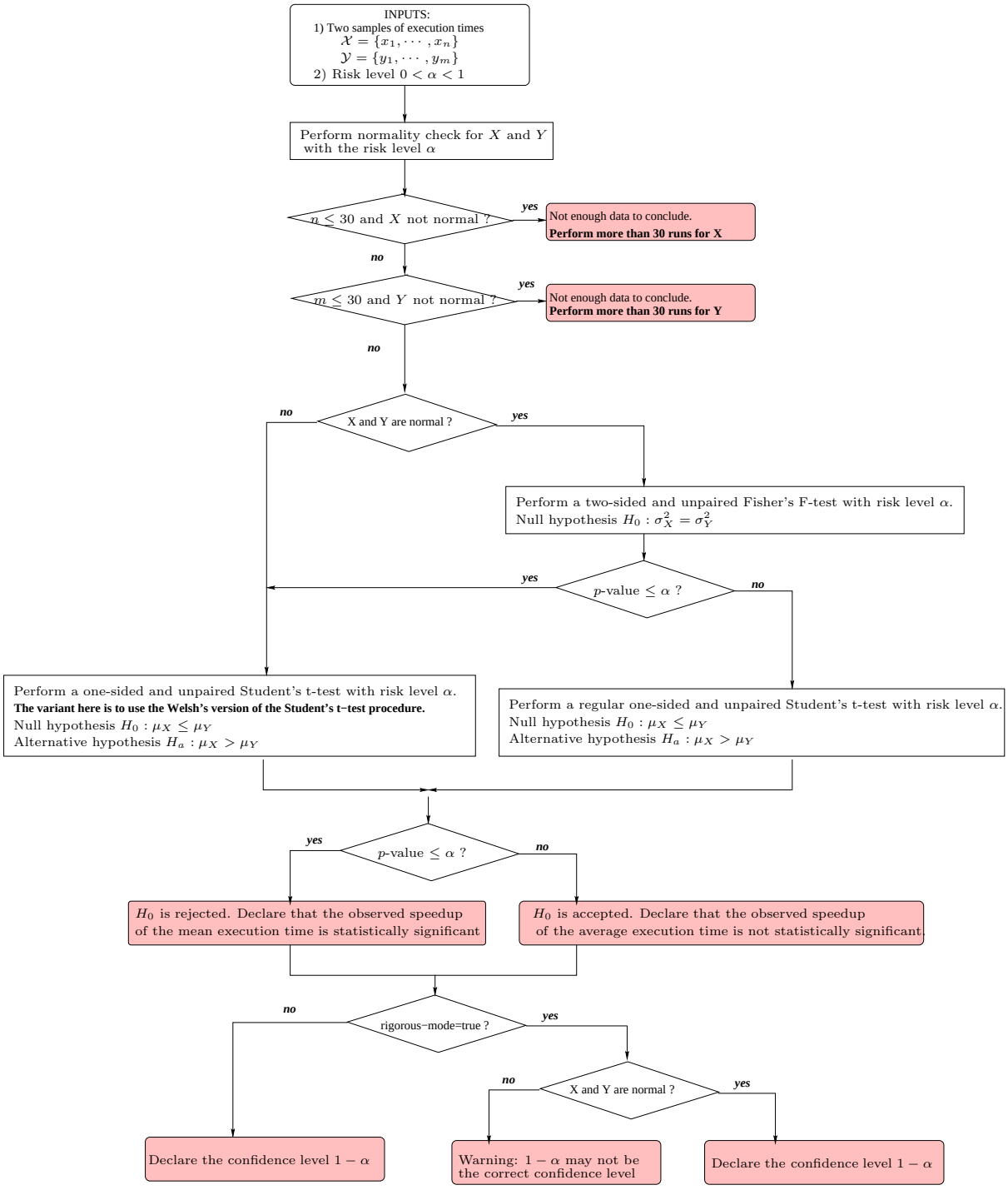

Figure 1: The Speedup Test for the Average Execution Time

We get p-value$= 0.9647 > \alpha$. *So* Y *can be assumed as normally distribution too. It is important to notice here that if the normality test fails for a program (*X *or* Y*), we must run it more than 30 times as illustrated in Figure 4.1.*

Since the two samples $\mathcal{X}$ *and* $\mathcal{Y}$ *can be assumed Gaussian, we now check whether they have similar variance with the Fisher's F-test.*

```
> var.test(X,Y)
F test to compare two variances
data:  X and Y
F = 1, num df = 4, denom df = 4, p-value = 1
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
 0.1041175 9.6045299
sample estimates:
ratio of variances
```

We can see that p-value$= 1$. *Since p-value*$> \alpha$, *we do not reject the equality of the variances. Consequently, we use the Student's t-test with this information.*

We can now continue with the Student's t-test to check if $\mathcal{C}'$ *is faster in average than* $\mathcal{C}$ *with a confidence level equal to* $1 - \alpha = 95\%$ *(we set the parameter* `var.equal=TRUE` *because the Fisher's F-test confirmed it).*

```
> t.test(X,Y, alternative="greater", var.equal = TRUE)
Two Sample t-test
data:  X and Y
t = 2.8238, df = 8, p-value = 0.01118
...
```

Here, the p-value$= 0.01118 \leq \alpha$. *Hence, we reject the null hypothesis that* $\mu_X \leq \mu_Y$ *with a risk* $\alpha = 5\%$. *This means that we accept the alternative hypothesis that* $\mu_X > \mu_Y$ *with a confidence level equal to* $1 - \alpha = 95\%$.

If the Fisher's F-test didn't conclude about the similarity between the variances, then we could have used the Welch's variant of the Student's t-test as follows:

```
> t.test(X,Y, alternative="greater")
Welch Two Sample t-test
data:  X and Y
t = 2.8238, df = 8, p-value = 0.01118
...
```

In this case, the p-value$= 0.0118$ *is the same than the previous one, but this is not necessarily the case in general. Finally, this example shows that the speedup* `spmean`$(\mathcal{C}, I) = \bar{X}/\bar{Y} = 1.95$ *has been demonstrated statistically significant with a confidence level of* 95%

The problem with the average execution time is its sensibility to outliers. Furthermore, the average is not always a good estimate of the observed execution time felt by the user. In addition, the test of Student has been proved only for Gaussian distributions, while it is rare in practice to observe them for program execution times [12]: the usage of the Student's t-test for non Gaussian distributions is admitted for large samples but the risk level is no longer guaranteed.

The median is generally preferable than the average for summarising the data into a single number. The next section shows how to check if the speedup of the median is statistically significant.

4.2 The speedup of the observed median execution time, as well as individual runs

This section presents the Wilcoxon-Mann-Whitney [8] test, a robust statistical test to check if the median execution time has been reduced or not after a program transformation. In addition, the statistical test we are presenting checks also if $\mathbb{P}[X > Y] > 1/2$, as is demonstrated in Appendix D: this is a very

good information for the real speedup felt by the user (the probability that a single random run of the optimised program is faster than a single random run of the initial program).

Contrary to the Student's t-test, the Wilcoxon-Mann-Whitney test does not assume any specific distribution for X and Y. The mathematical model (page 70 in [8]) imposes that the distributions of X and Y differ only by a location shift Δ, in other words that

$$F_Y(t) = \mathbb{P}[Y \leq t] = F_X(t + \Delta) = \mathbb{P}[X \leq t + \Delta] \quad (\forall t)$$

Under this model (known as the *location model*), the location shift equals $\Delta = \text{med}(X) - \text{med}(Y)$ (as well as $\Delta = \mu_X - \mu_Y$ in fact) and X and Y consequently do not differ in dispersion. If this constraint is not satisfied, then as admitted for the Student's t-test, the Wilcoxon-Mann-Whitney test can still be used for large samples in practice but the announced risk level may not be preserved. However, two advantages of this model is that the normality is not needed any more and that assumptions on the sign of Δ can be readily interpreted in terms of $\mathbb{P}[X > Y]$.

In order to check if X and Y satisfy the mathematical model of the Wilcoxon-Mann-Whitney test, a possibility is to use the Kolmogorov-Smirnov's two sample test ([20]) as described below. "

Using the test of Kolmogorov-Smirnov first: The object is to test the null hypothesis H_0 of equality of the distributions of the variables $X - \overline{\text{med}}(X)$ and $Y - \overline{\text{med}}(Y)$, using the Kolmogorov-Smirnov two-sample test applied to the observations $x_i - \overline{\text{med}}(X)$ and $y_j - \overline{\text{med}}(Y)$. The Kolmogorov-Smirnov's test computes a p-value : if p-value$\leq \alpha$, then H_0 is rejected with a risk level α. That is, X and Y do not satisfy the mathematical model needed by the Wilcoxon-Mann-Whitney test. However, as said before, we can still use the test in practice for sufficiently large samples but the risk level may not be preserved [8].

Using the test of Wilcoxon-Mann-Whitney: As done previously with the Student's t-test for comparing between two averages, we want here to check whether the median of X is greater than the median of Y, and if $\mathbb{P}[X > Y] > \frac{1}{2}$. This amounts to use the one-sided variant of the test of Wilcoxon-Mann-Whitney. In addition, since the observation x_i from X does not correspond to an observation y_j from Y, we use the unpaired version of the test.

We set the null hypothesis H_0 of Wilcoxon-Mann-Whitney's test as $F_X \geq F_Y$, so the alternative hypothesis is $H_a : F_X < F_Y$. As a matter of fact, $F_X < F_Y$ means that X tends to be greater than Y. Note in addition that, under the location shift model, H_a is equivalent to the fact that the location shift Δ is > 0.

The Wilcoxon-Mann-Whitney test computes a p-value. If p-value$\leq \alpha$, then H_0 is rejected. That is, we admit H_a with a confidence level $1 - \alpha$: $F_X > F_Y$. This amounts to declaring that the observed speedup of the median executions times is statistically significant, $\text{med}(X) > \text{med}(Y)$ with a confidence level $1 - \alpha$, and $\mathbb{P}[X > Y] > \frac{1}{2}$. If the null hypothesis is not rejected, then the observed speedup of the median is not considered to be statistically significant.

Figure 4.2 illustrates the Speedup-Test protocol for the median execution time.

Example (done with R) 2. *Let consider the application* `bzip2` *from the SPEC-CPU2006 benchmark suite. We generated two binary codes for this application using* `gcc` *version 4.3 under linux. A first binary was generated without optimisation flag of gcc. A second binary has been generated with* `gcc -O3`*. We executed each of the two binaries 32 times using the reference data input. The measures have been done on an isolated linux workstation with the lowest possible background workload. X is the set of 32 observed execution times of the first binary code, and Y is the 32 observed execution times of the binary code generated with* `gcc -O3`*. The observed speedup of the median is* `spmedian`$(bzip2, ref) = \frac{\overline{med}(X)}{\overline{med}(Y)} = 2.04$*. In order to demonstrate that this speedup is statistically significant with a risk level $\alpha = 5\%$, we proceed as follows with* `R`*. First, we load the data into two vectors X and Y.*

```
> X <- scan ("SPECCPU2006/inputdata_ref/401.bzip2.00.pl.csv.1")
> Y <- scan("SPECCPU2006/inputdata_ref/401.bzip2.01.pl.csv.2")
```

We have to check whether $X - med(X)$ and $Y - med(Y)$ have the same distribution using the Kolmogorov-Smirnov's test.

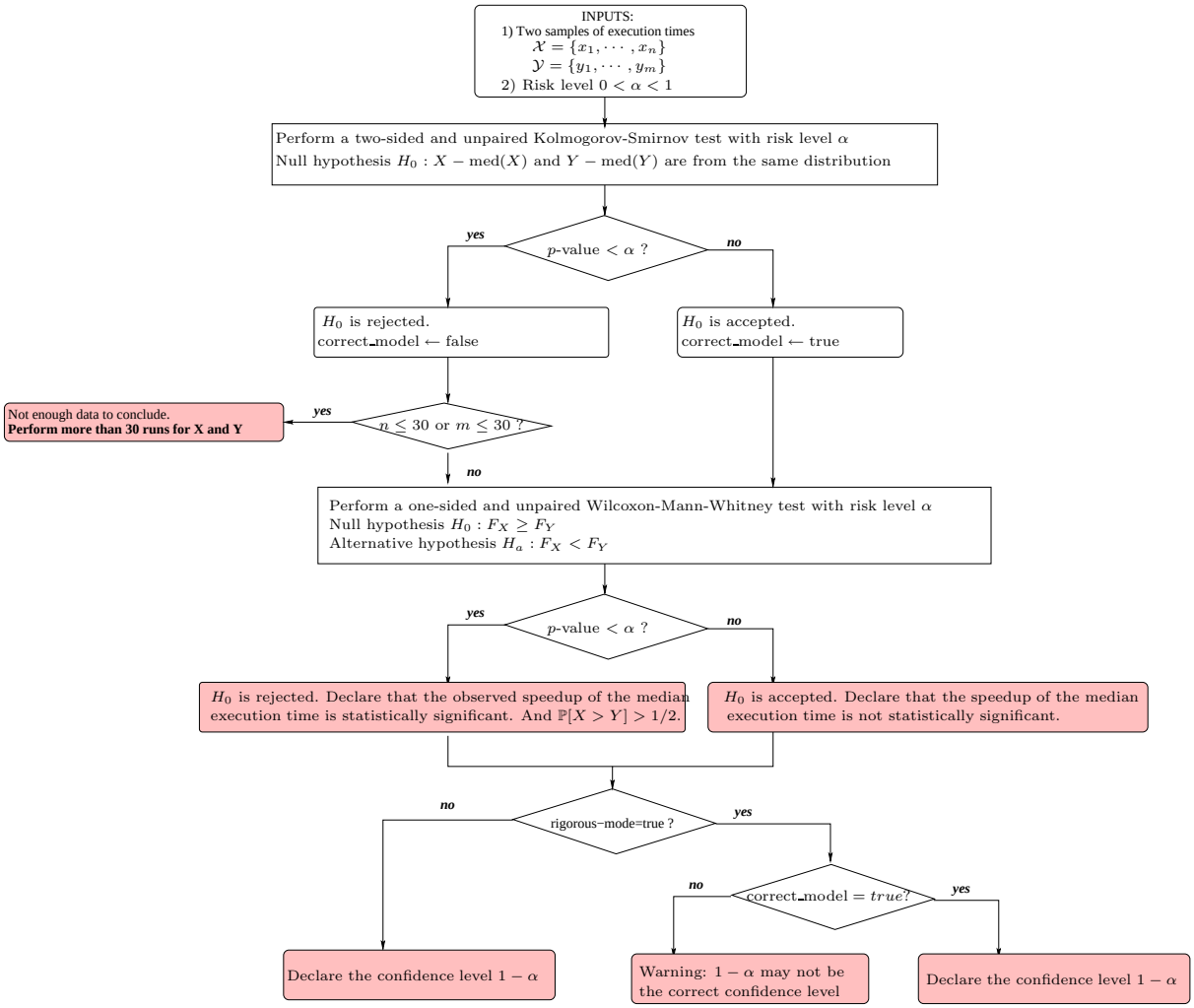

Figure 2: The Speedup Test for the Median Execution Time

```
>ks.test(X-median(X),Y-median(Y))
...
Two-sample Kolmogorov-Smirnov test
...
data:  X - median(X) and Y - median(Y)
D = 0.2812, p-value = 0.1590
```

The computed p-value$= 0.159 > \alpha = 0.05$*, so we do not reject the null hypothesis of the test. This means that we can admit that* $X - med(X)$ *and* $Y - med(Y)$ *fit in the location shift model. Consequently,* X *and* Y *satisfy the constraints of the shoft location model required by the Wilcoxon-Mann-Whitney's test. If the Kolmogorov-Smirnov's test had rejected the null hypothesis, then we could still have used the Wilcoxon-Mann-Whitney's test (provided large samples* $\mathcal{X}$ *and* $\mathcal{Y}$*) but the risk level might not have been preserved [8]. In the following* `R` *command, the option* `alternative="greater"` *means that* $\Delta > 0$ *where* $X = Y + \Delta$*.*

```
> wilcox.test(X, Y, alternative="greater")
...
Wilcoxon rank sum test with continuity correction
...
data:  X and Y
W = 1024, p-value = 3.245e-12
```

The computed p-value$< \alpha$ *concludes that we reject the null hypothesis. So we accept the alternative one, which is* $X = Y + \Delta$ *and* $\Delta > 0$*. Consequently, the observed speedup of the median execution time is statistically significant with a confidence level* $1 - \alpha = 95\%$*. In addition, the Wilcoxon-Mann-Whitney's test allows to conclude that* $\mathbb{P}[X > Y] > 1/2$*, which means that single random run* Y *has more chance to be faster than a single random* X*.*

5 Measuring the confidence interval of the proportion of accelerated benchmarks

Computing the overall performance gain or speedup for a sample of b programs does not allow to estimate the quality nor the efficiency of the code optimisation technique. In fact, within the b programs, only a fraction of a benchmarks have got a speedup, and $b-a$ programs got a slowdown.

In probability theory, we can study the random event {`The program is accelerated with the code optimisation under study`}. This event has two possible values, `true` or `false`. So it can be represented by a Bernoulli (binomial) variable. In order to make correct statistics, it is very important that the initial set of b benchmarks must be selected randomly from a huge database of representative benchmarks. If the set of b benchmarks are selected manually and not randomly, then there is a bias in the sample and the statistics we compute in this section are wrong.

If we select randomly a sample of b representative programs as a sample of benchmarks, we can measure the chance of getting the fraction of accelerated programs as $\frac{a}{b}$. The higher this proportion is, the better the quality of the code optimisation would be. In fact, we want to evaluate whether the code optimisation technique is beneficial for a large fraction of programs. The proportion $C=\frac{a}{b}$ has been observed on a sample of b programs only. There are many techniques for estimating the confidence interval for $\frac{\mu_a}{b}$ (with a risk level α). The simplest and most commonly used formula relies on approximating the binomial distribution with a normal distribution. In this situation, the confidence interval is given by the equation $C \mp r$, where $r = z_{1-\alpha/2} \times \sqrt{\frac{C(1-C)}{b}}$. In other words, the confidence interval of the proportion is equal to $[C-r, C+r]$. Here, $z_{1-\alpha/2}$ represents the value of the quartile of order $1-\alpha/2$ of the standard normal distribution ($\mathbb{P}[N(0,1) > z] = \frac{\alpha}{2}$). This value is available in a known precomputed table and in many softwares (table A.2 in [15]). We should notice that the previous formula of the confidence interval of the proportion C is accurate only when the value of C are not too close from 0 or 1. A frequently cited rule of thumb is that the normal approximation works well as long as $a - \frac{a^2}{b} > 5$, as indicated for instance in [6] (section 2.7.2). However, in a recent contribution [1], that condition was discussed and criticised. The general subject of choosing appropriate sample size which ensures an accurate normal approximation, was discussed in chapter VII 4, example (h) of the reference book [4].

When the approximation of the binomial distribution with a normal one is not accurate, other techniques may be used, that will not be presented here. The `R` software has an implemented function that computes the confidence interval of a proportion based on the normal approximation of a binomial distribution, see the example below.

Example (done with R) 3. *Having $b = 30$ benchmarks selected randomly from a huge set of representative benchmarks, we obtained a speedup on only $a = 17$ cases. We want to compute the confidence interval for the proportion C=17/30=0.5666 with a risk level equal to $\alpha = 0.1 = 10\%$. We easily estimate the confidence interval of C using the R software as follows.*

```
> prop.test(17, 30, conf.level=0.90)
...
90 percent confidence interval:
 0.4027157 0.7184049
...
```

Since $a - \frac{a^2}{b} = 17 - \frac{17^2}{30} = 7.37 > 5$, the computed confidence interval of the proportion is accurate. Note that this confidence interval is invalid if the initial set of $b = 30$ benchmarks was not randomly selected among a huge number of representative benchmarks.

The above test allows us to say that we have 90% of chance that the proportion of accelerated programs is between 40.27% and 71.87%. If this interval is too wide for the purpose of the study, we can reduce the confidence level as a first straightforward solution. For instance, if I consider $\alpha = 50\%$, the confidence interval of the proportion becomes $[49.84\%, 64.23\%]$.

The next formula[15] gives the minimal number b of benchmarks to be selected randomly if we want to estimate the proportion confidence interval with a precision equal to $r\%$ with a risk level α:

$$n \geq (z_{1-\alpha/2})^2 \times \frac{C(1-C)}{r^2}$$

Example (done with R) 4. *In the previous example, we have got an initial proportion equal to* $C = 17/30 = 0.5666$. *If we want to estimate the confidence interval with a precision equal to 5% with a risk level of 5%, we put* $r = 0.05$ *and we read in the quartiles tables* $z_{1-0.05/2} = z_{0.975} = 1.960$. *The minimal number of benchmarks to observe is then equal to:* $b \geq 1.960^2 \times \frac{0.566 \times (1-0.566)}{0.05^2} = 377.46$. *We need to randomly select 378 benchmarks in order to assert that we have 95% of chances that the proportions of accelerated programs are in the interval* $0.566 \mp 5\%$.

6 The Speedup-Test software

All the protocols described in this document has been implemented and automated in a script software called `SpeedupTest`. The user has only to enter its observations data and write a configuration file for the parameters of the statistics. The software checks if the observed speedups are statistically significant and makes a full report.

More precisely, the software determines whether there is a statistically observable speedup between two programs versions P_i (initial program) and P'_i (transformed program).

The observed *speedup ratio* is computed by the formula

$$\rho_i = \frac{\texttt{ExecutionTime}(P_i, Input)}{\texttt{ExecutionTime}(P'_i, Input)}$$

Execution time measures are presented in Comma-Separated Values files, where each file corresponds to a benchmark and is composed of two columns of data, which correspond to observed time. The *execution time* of a program P is a chosen value within a set of observed execution times: the chosen value can be either the minimum time, or the mean time, or the median time.

The *gain factor* of the entire set of programs is previously defined as:

$$\begin{aligned} G &= \frac{\sum_{i=1}^{i=b} w(P_i)(\texttt{ExecutionTime}(P_i, Input) - \texttt{ExecutionTime}(P'_i, Input))}{\sum_{i=1}^{i=nb} w(P_i)\texttt{ExecutionTime}(P_i, Input)} \\ &= 1 - \frac{\sum_{i=1}^{i=b} w(P_i)\texttt{ExecutionTime}(P'_i, Input)}{\sum_{i=1}^{i=b} w(P_i)\texttt{ExecutionTime}(P_i, Input)} \end{aligned}$$

where $w(P_i)$ is the *weight* of program P_i.

The weight of program P_i can be any positive number. The only constraint is that the *total weight* $\sum_{i=1}^{i=n} \epsilon_i\, w(P_i)$ is not null. The execution time of a program (with a fixed input) can be set as the minimum, average or the median of the observed executions times.

If a is the number of benchmarks where a speedup can be observed, then the proportion of *accelerated* benchmarks is $C = \frac{a}{b}$.

As previously shown in Section 5, it is possible to estimate the number of needed benchmarks b, in order to measure a confidence interval of this proportion with a desired precision r_0.

6.1 Prerequisites and installation

The present software package needs the R software [14] to be installed and executable in the `PATH`.

The present software package is composed of a shell script `SpeedUpTest.sh` and a R script `SpeedUpTest.R`.

These two files may be copied at any place, provided they are put both in the same directory. In the sequel, we assume that both these scripts are located in the directory `/usr/local/SpeedUp`.

Execution rights must be granted to the shell script, e.g. by doing

```
chmod +x /usr/local/SpeedUp/SpeedUpTest.sh
```

It may be convenient to symbolically link the shell script to a directory listed by the `PATH` environment variable, e.g. by doing

```
ln -s /usr/local/SpeedUp/SpeedUpTest.sh /usr/local/bin
```

6.2 Usage

The shell script may be invoked as follows:

```
SpeedUpTest.sh config.csv
               [--conf-level #value]
               [--weight (custom|equal|fraction)]
               [--precision #value]
               [-o report-file]
```

In the sequel, we explain the precise meaning of these flags.

6.2.1 Configuring the input of the statistical analysis

The file `config.csv` is a Comma-Separated Values file that describes the set of benchmarks to process. Data must be composed of exactly five fields:

1. the `Name` field refers to a string that is the name of the benchmark;
2. the `Sample1` field refers to the file name containing the measured execution times of the benchmark for the first program; The path of these file names are relative to the current path of the shell command;
3. the `Sample2` field refers to the file name containing the measured execution times of the benchmark for the second program; The path of these file names are relative to the current path of the shell command;
4. the `ConfLevel` field refers to a value (the confidence level) between 0 and 1;
5. the `Coef` field refers to a positive value used to compute the weight of the benchmark. In other words, the weights are the normalised coefficients. The relationship between the coefficients and the weights of a program P_i is simply $W(P_i) = \frac{Coef(P_i)}{\sum_j Coef(P_j)}$

Only the three first fields (`Name`, `Sample1`, `Sample2`) are mandatory. The other two may be left empty, or take the `NA` (Not Available) value. If left empty, the comma must still be present for validity of csv file format.

A valid configuration file is shown below.

```
Name,Sample1,Sample2,ConfLevel,Coef
"First benchmark","bench/bench1.1","bench/bench1.2",0.9,2
"Second benchmark","bench/bench2.1","bench/bench2.2",0.8,1.5
"Third benchmark","bench/bench3.1","bench/bench3.2",0.95,1
```

An other valid configuration file, that omits some values, is shown below.

```
Name,Sample1,Sample2,ConfLevel,Coef
"First benchmark","bench/bench1.1","bench/bench1.2",NA,
"Second benchmark","bench/bench2.1","bench/bench2.2",,
"Third benchmark","bench/bench3.1","bench/bench3.2",0.95,
```

6.2.2 Setting the confidence level

The confidence level to use is freely adjustable for each benchmark. The user simply needs to specify the desired value in the configuration file. Note that the confidence level is $1 - \alpha$, where α is the risk level used as parameter in all our statistical tests.

However, when the the value of a confidence level is omitted or not valid (i.e. not between 0 and 1), a default confidence level may be used instead. This is precisely the purpose of the option flag `--conf-level` which takes in addition a value between 0 and 1.

If no default value is specified, then our software tries to find the best possible confidence level that allows to declare a statistically significant speedup. Note that the value of the confidence level might be very low if no effective speedup can be statistically declared. To avoid such situation, if no speedup can be found with a confidence level greater than 50%, then a warning is emitted and the statistical test fails.

6.2.3 Setting the weights of the benchmarks

The weight of benchmarks can be individually set in the configuration file. However, it is possible to override these thanks to:

- `--weight equal` to set all benchmarks weights to be equal;

- `--weight fraction` to set benchmarks weights using the formula

$$w(P_i) = \frac{\texttt{ExecutionTime}(P_i, Input)}{\sum_{i=1}^{i=b} \texttt{ExecutionTime}(P_i, Input)}$$

- `--weight custom` to use custom weights of the configuration file (by default).

6.2.4 Confidence interval of the proportion of accelerated benchmarks

The confidence level used to compute the confidence interval of the proportion $\frac{a}{b}$ (proportion of the benchmarks with speedups vs. all benchmarks) is either the one specified by the command line, or if none is given, then the default value of 95% is taken.

In order to estimate the minimal number of benchmarks that is necessary to measure the proportion $\frac{a}{b}$ with a precision r, we must give the value of this r parameter by using the option `--precision #value`

By default, the precision is equal to 5%, corresponding to `--precision0.05`.

6.2.5 Setting the report file name

The statistical analysis produces four separate files: the status file, the results file, the warnings file and the report file.

The status file logs the possible errors that might have happened during the analysis. If the analysis terminates successfully, it only mentions the elapsed time (of the R process).

The results file is a CSV file which is composed of several columns. These columns are:

1. `Name` which is the descriptive name of the benchmark;
2. `SpeedupMin` which is the speedup ratio, using the minimum execution time of P_i for $\texttt{ExecutionTime}(P_i, Input)$
3. `SpeedupMean` which is the speedup ratio, using the mean execution time of P_i for $\texttt{ExecutionTime}(P_i, Input)$
4. `IsMeanSignificant` indicates whether the observed speedup for the mean execution time is statistically significant (the value of this column is TRUE or FALSE or NA).
5. `MeanConfLevel` is the confidence level used for the statistical test of the mean speedup ratio. Its value may be NA if there is a problem with the input data.
6. `SpeedupMedian` which is the speedup ratio, using the median execution time of P_i for $\texttt{ExecutionTime}(P_i, Input)$
7. `IsMedianSignificant` indicates whether the observed speedup for the median execution time is statistically significant (the value of this column is TRUE or FALSE or NA).
8. `MedianConfLevel` is the confidence level used for the statistical test of the median speedup ratio. Its value may be NA if there is a problem with the input data.
9. `CoefMin` which is the actual coefficient used to compute the weight of the benchmark, in the gain factor formula, when $\texttt{ExecutionTime}(P_i, Input)$ is the minimum execution time of P_i
10. `CoefMean` which is the actual coefficient used to compute the weight of the benchmark, in the gain factor formula, when $\texttt{ExecutionTime}(P_i, Input)$ is the mean execution time of P_i
11. `CoefMedian` which is the actual coefficient used to compute the weight of the benchmark, in the gain factor formula, when $\texttt{ExecutionTime}(P_i, Input)$ is the median execution time of P_i

The warnings file contains all the warnings that accompany the statistical tests. The warnings can be the followings:

- `File 'xxx' is not readable`

  A sample file is not readable. The benchmark will be ignored.

- `Cannot process benchmark: samples unavailable.` This happens when at least one of the sample file is not readable.

- `Sample? too small for applying the Student's t-test (speedup of the mean). Please do more than 30 observations of the executions times.`

  One of the sample do not contain enough data to perform the statistical Student's t-test needed to analyse the average execution time.

- `Sample2 data are not normally distributed. The indicated confidence level for the speedup of the average execution time may not be accurate.`

  The statistical Student's t-test requires that the two samples follow gaussian distributions. When this is not the case but when the sample contains enough data (more than 30), then the statistical test can be used but the confidence level cannot be guaranteed.

- `Sample? too small for applying the Wilcoxon-Mann-Whitney's test (speedup of the median). Please do more than 30 observations of the executions times.`

  One of the sample do not contain enough data to perform the statistical Wilcoxon-Mann-Whitney's test needed to analyse the median execution time.

- `The two samples do not fit the location shift model. The indicated confidence level for the speedup of the median execution time may not be accurate.`

  The statistical Wilcoxon-Mann-Whitney's test requires that the two samples follow the same distribution upto a translation. When this is not the case but when the samples contain enough data (more than 30), then the statistical test can be used but the confidence level cannot be guaranteed.

- `Unable to find a confidence level greater than 50% to guarantee the statistical significance of mean speedup.`

  It was not possible to find a confidence level greater than 50% that guaranteed statistical significance of mean comparison (speedup of the average). It could simply say that no speedup occurs at all or that there is a problem with one (or the two) samples.

- `Unable to find a confidence level greater than 50% to guarantee the statistical significance of median speedup.`

  It was not possible to find a confidence level greater than 50% that guaranteed statistical significance of median comparison (speedup of the median). It could simply say that no speedup occurs at all or that there is a problem with one (or the two) samples.

The report file contains a precise report of the statistical analysis.

By default, if the input csv configuration file is named `inputfilename`, then the status file is named `inputfilename.status`, the results file is named `inputfilename.out`, the report file is named `inputfilename.report` and the warnings file is named `inputfilename.warning`.

To change the name of the used prefix use to generate these files, use

- `-o outputprefix`

6.3 Example

6.3.1 Collected data

In the following, we pursue the analysis of four benchmarks. The observed executions times of four are listed in four × two CSV files (initial an optimised version) named `bench1.data.1`, `bench1.data.2`, `bench2.data.1`, `bench2.data.2`, `bench3.data.1`, `bench3.data.2` and `bench4.data.1`, `bench4.data.2`.

The first benchmark is composed of 2×5 measures (initial an optimised version).

```
> cat bench1.data.1
2.02
2.25
```

```
2.30
2.251
2.01
> cat bench1.data.2
1.02
2.05
2.30
2.071
1.05
```

The second benchmark is composed of 2×6 measures (initial an optimised version).

```
> cat bench2.data.1
2.799
2.046
1.259
1.877
2.244
> cat bench2.data.2
1.046
0.259
0.877
1.244
1.799
```

The third benchmark is composed of respectively 15 and 20 measures (initial an optimised version). Observe in particular that the number of measures for the two programs to compare need not to be the same.

```
> cat bench3.data.1
6.512692
5.547728
4.171278
5.748114
6.188147
4.860546
6.393239
5.862367
5.724749
7.769651
6.455157
6.975127
5.331494
6.779595
4.839683
> cat bench3.data.2
4.556838
5.491279
5.708276
5.204911
4.454981
5.059760
5.440053
4.780246
4.363734
5.782297
5.195786
```

```
5.627607
6.114562
6.552509
3.055505
4.037513
5.445448
3.665237
6.965091
4.396594
```

The fourth benchmark is composed of respectively 4 and 8 measures (initial an optimised version).

```
> cat bench4.data.1
7.308153
6.891170
6.102855
6.472642
> cat bench4.data.2
6.571750
5.514734
5.705132
7.051386
8.007863
4.187613
6.124584
4.995708
```

The configuration file, named `bench.cfg`, is shown below.

```
> cat bench.cfg
Name,Sample1,Sample2,ConfLevel,Coef
"First sample","bench1.data.1","bench1.data.2",NA,
"Second sample","bench2.data.1","bench2.data.2",NA,NA
"Third sample","bench3.data.1","bench3.data.2",,NA
"Fourth sample","bench4.data.1","bench4.data.2",,
```

Remark that neither confidence levels nor weights are set. So by default the weights are considered equal for all benchmarks unless the command line option specifies something else. For the confidence levels, the default behaviour of Speedup-Test software is that it search for the highest confidence level $> 50\%$ (lowest risk level $< 50\%$) that allows to declare a statistically significant speedup. This behaviour can be modified by the command line option that allows to specify a global confidence level for all the benchmarks.

6.3.2 Carrying on the Speedup-Test on the data

We analyse this set of data thanks to the following command.

```
> SpeedUpTest.sh bench.cfg
Analysis report of  ./bench.cfg

Overall gain (ExecutionTime=min) = 0.371
Overall speedup (ExecutionTime=min) = 1.589

Overall gain (ExecutionTime=mean) = 0.178
Overall speedup (ExecutionTime=mean) = 1.216

Overall gain (ExecutionTime=median) = 0.156
Overall speedup (ExecutionTime=median) = 1.185
```

The above messages print the overall gains and speedups depending on the chosen function to summarise the execution time of a program. Concerning the proportion of the accelerated programs, it is printed in the message below. A program is considered as accelerated if the speedup-test succeeds in declaring a statistically significance of speedup of its average or median execution times, as detailed in Section 4. So we have two sorts of proportions, depending if we consider the average of the median execution time.

```
The observed proportion of accelerated benchmarks (speedup of the mean) a/b =
3/4 = 0.75
The confidence level for computing proportion confidence interval is 0.95.
Proportion confidence interval (speedup of the mean) = [0.219; 0.987]
Warning: this confidence interval of the proportion may not be accurate because
the validity  condition {a(1-a/b)>5} is not satisfied.

The minimal needed number of randomly selected benchmarks is 289 (in order to
have a precision r=0.05).

Remark: The computed confidence interval of the proportion is invalid if b the
experimented  set of benchmarks is not randomly selected among a huge number of
representative benchmarks.

The observed proportion of accelerated benchmarks (speedup of the median) a/b =
4/4 = 1.
The confidence level for computing proportion confidence interval is 0.95.
Proportion confidence interval (speedup of the median) = [0.396; 1].
Warning: this confidence interval of the proportion may not be accurate because
the validity  condition {a(1-a/b)>5} is not satisfied.

Remark: The computed confidence interval of the proportion is invalid if b the
experimented set of benchmarks is not randomly selected among a huge number of
representative benchmarks.
```

We observe that there is a statistically significant speedup for 3 out of the 4 samples for the mean and 4 out of 4 for the median. The Speedup-Test computes a confidence interval for each of the proportions $\frac{a}{b} = \frac{3}{4}$ and $\frac{a}{b} = \frac{4}{4}$. However the printed warning clearly says that the confidence intervals may not be accurate because the condition $a \times (1 - a/b) > 5$ is not satisfied, see Sectio, 5 for more details. We recall that the confidence intervals of the proportions are invalid if the initial set of b benchmarks is not *randomly* selected among a huge set of benchmarks. In other words, a manual selection of a set of experimented benchmarks (such as from SPEC family or other public benchmarks) is an invalid approach to calculate the confidence interval of the proportion $\frac{a}{b}$.

Now, let us have now a look at the warnings that occurred during the Speedup-Test analysis.

```
Warnings regarding analysis of ./bench.cfg
First sample :
  Sample1 too small for applying the Student's t-test (speedup of the mean).
Please do more than 30 observations of the executions times.
  Sample1 too small for applying the Student's t-test (speedup of the mean).
Please do more than 30 observations of the executions times.
  Unable to find a confidence level greater than 50% to guarantee the
statistical significance of mean speedup.
3 warning(s).
```

These warnings tell us that there are not enough data for the first benchmark in order to analyse the statistical significance of the speedup of the mean execution time. Probably because the executions times of the first sample are not normally distributed, so the Student's t-test require more than 30 values to stay robust.

6.4 Licence: Copyright UVSQ (2010)

7 Discussion and conclusion

Program performance evaluation and their optimisation techniques suffer from the non reproducibility of published results. It is of course very difficult to reproduce exactly the experimental environment since we do not always know all the details or factors influencing it [17]. This document treats a part of the problem by defining a rigorous statistical protocol allowing to consider the variations of program execution times if we set the execution environment. The variation of program execution times is not a chaotic phenomena to neglect or to smooth; we should keep it under control and incorporate it inside the statistics. This would allow us to assert with a certain confidence level that the performance data we report are reproducible under similar experimental environment. The statistical protocol that we propose to the community in this article is called the Speedup-Test and is based on clean statistics as described in [6, 8, 13].

Compared to [11, 15], the Speedup-Test protocol analyses the median execution time in addition to the average. Contrary to the average, the median is a better performance metric because it is not sensitive to outliers and is more appropriate for skewed distributions. Summarising the observed executions times of a program with their median allows to evaluate the chance to have a faster execution time if we do a single run of the application. Such performance metric is closer to the feeling of the users in general. Consequently, the Speedup-Test protocole is more rigorous then the protocoles described [11, 15] based on the average execution times. Additionaly, the Speedup-Test protocole is more cautious than [11, 15] becauses it checks the hypothesis on the data distributions before applying statistical tests.

The Speedup-Test protocol rigorously analyses the distribution of the observed executions times. For declaring a speedup for the average execution time, we rely on the Student's t-test under the condition that X and Y follow a Gaussian distribution (tested with Shapiro-Wilk's test). If not, using the Student's t-test is admitted but the computed risk level α may still be inaccurate if the underlying distributions of X and Y are too far from being normally distributed. For declaring a speedup for the median execution time, we rely on the Wilcoxon-Mann-Whitney's test. Contrary to the Student's t-test, the Wilcoxon-Mann-Whitney's test does not assume any specific distribution of the data, except that it requires that X and Y differ only by a shift location (that can be tested with the Kolmogorov-Smirnov's test).

Using simulators instead of real executions provide reproducible results, since simulators are deterministic: usually, simulating a program multiple times always produce the same performance numbers. This document assumes that the observations have been done on the physical machine not by simulation. If the physical machine does not exist, the observations based on simulation cannot be studied exactly with the methods described in this article. The study should more be concentrated on the statistical quality of the simulator. As far as we know, it does not exist yet a simulator that has been rigorously validated by statistics. Usual error ratios reported by simulators are not sufficient alone to judge about their quality.

We conclude with a short discussion about the risk level we should use in this sort of statistical study. Indeed, there is not a unique answer to this crucial question. In each context of code optimisation we may be asked to be more or less confident in our statistics. In the case of hard real time applications, the risk level must be low enough (less than 5% for instance). In the case of soft real time applications (multimedia, mobile phone, GPS, etc.), the risk level can be less than 10%. In the case of desktop applications, the risk level may not be necessarily too low. In order to make a fair report of a statistical analysis, we advise to make public all the experimental date and the risk levels used for the statistical tests.

References

[1] Lawrence D Brown, Tony Cai, and Anirban DasGupta. Interval Estimation for a Binomial Proportion. *Statistical Science*, 16(2):101–133, 2001. With comments and a rejoinder by the authors.

[2] Pierre-André Cornillon, Arnaud Guyader, Nicolas Jégou François Husson, Julie Josse, Maella Kloareg, Éric Matzner-Lober, and Laurent Rouviére. *Statistiques avec R.* Société Française de statistique, Presses universitaires de Rennes edition.

[3] Daniel Gracia Pérez and Gilles Mouchard and Olivier Temam. MicroLib: A Case for the Quantitative Comparison of Micro-Architecture Mechanisms. In *MICRO*, 2004.

[4] William Feller. *An introduction to probability theory and its applications*, volume 1. John Wiley and Sons, Inc., 1968. Third edition.

[5] Andy Georges, Dries Buytaert, and Lieven Eeckhout. Statistically rigorous Java performance evaluation. In *Proceedings of the Twenty-Second ACM SIGPLAN Conference on OOPSLA*, ACM SIGPLAN Notices 42(10), pages 57–76, Montréal, Canada, October 2007.

[6] Gilbert Saporta. *Probabilités, analyse des données et statistique.* Editions Technip, Paris, France, 1990. ISBN 978-2-7108-0814-5.

[7] Mann H.B. and Whitney D.R. On a test of whether one of two random variables is stochastically larger than the other. *Ann. of Math. Statistics*, 18(1):50–60, 1947.

[8] Myles Hollanderand and Douglas A. Wolfe. *Nonparametric Statistical Methods.* Wiley-Interscience, 1973. ISBN: 0-471-40635-X.

[9] John L. Hennessy and David A. Patterson and David Goldberg . *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann, 2002. ISBN-13: 978-1558605961.

[10] Keith Knight. *Mathematical statistics.* Chapman & Hall/CRC Press, 2000.

[11] David J. Lilja. *Measuring Computer Performance: A Practitioner's Guide.* Cambridge University Press, 2000.

[12] Abdelhafid Mazouz, Sid-Ahmed-Ali Touati, and Denis Barthou. Study of Variations of Native Program Execution Times on Multi-Core Architectures. In *International Workshop on Multi-Core Computing Systems (MuCoCoS).* IEEE, Krakow, Poland, February 2010.

[13] Peter J. Brockwell and Richard A. Davis. *Introduction to Time Series and Forecasting.* Springer, 2002. ISBN-13: 978-0387953519.

[14] R Development Core Team. *R: A Language and Environment for Statistical Computing.* 2008.

[15] Raj Jain. *The Art of Computer Systems Performance Analysis : Techniques for Experimental Design, Measurement, Simulation, and Modelling.* John Wiley and Sons, inc., New York, 1991.

[16] Sid-Ahmed-Ali Touati. Towards a Statistical Methodology to Evaluate Program Speedups and their Optimisation Techniques. Technical report, PRiSM laboratory. University of Versailles Saint-Quentin en Yvelines, January 2009. Technical report.

[17] Todd Mytkowicz et Amer Diwan et Peter F. Sweeney et Mathias Hauswirth. Producing wrong data without doing anything obviously wrong! In *ASPLOS*, 2009.

[18] D Van Dantzig. On the consistency and the power of wilcoxon's two-sample test. In *Koninklijke Nederlandse Akademie Van Wetenschappen, Proceedings, Ser. A*, volume 54, pages 1–9, 1951.

[19] A. W. van der Vaart. *Asymptotic statistics.* 2000. ISBN 0-521-78450-6, 0-521-49603-9, 978-0-521-78450-4, 978-0-521-49603-2.

[20] W. J. Conover. *Practical Nonparametric Statistics.* John Wiley, New York, 1971.

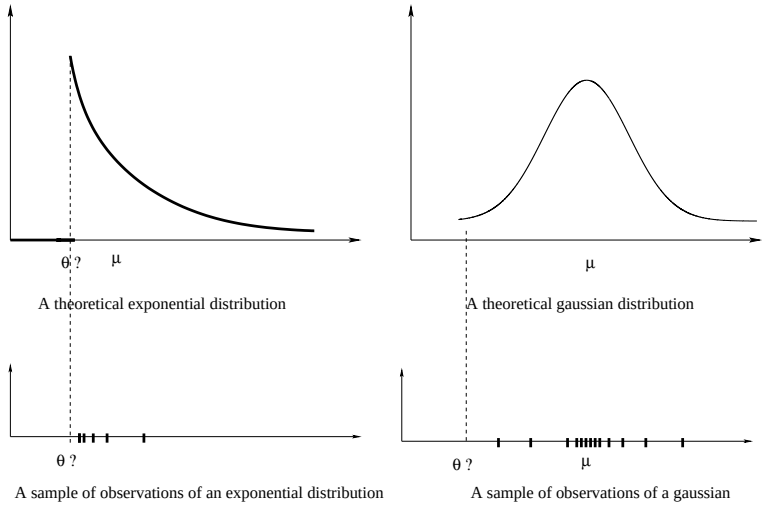

Figure 3: The sample min is a not necessarily a good estimation of the theoretical min

A Why the observed minimal execution time is not necessarily a good statistical estimation of program performances ?

Considering the minimum value of the n observed executions times is sometimes used but can be discussed:

- Nothing guarantees that this minimum execution time is an *ideal* execution of the program.
- Nothing guarantees that the minimum execution time over multiple runs represents the run with the least noise.
- Nothing guarantees that this minimum execution time is a consequence of the optimisation technique under study. Maybe this minimum execution time is an accident, or a consequence of dynamic voltage scaling, or anything else.
- If this minimal execution time is a rare event, all the statistics based on the minimum describe rare speedups. So, they become non-reproducible easily.

In addition to the above arguments, there is a mathematical argument against using the min. Indeed, contrary to the sample average or the sample median, the sample minimum does not follow a normal distribution necessarily. That is, the sample minimum does not necessarily converge quickly towards its theoretical value. The variance of the sample minimum may be pretty high for an arbitrary population. Formally, if θ is the theoretical minimal execution time, then the sample $\min_i x_i$ may be far from θ, all depends on the distribution function of X. As illustration, see Figure 3 for two cases of distributions, explained below:

Case of exponential distribution function shifted by θ If X follows an exponential distribution function shifted by θ, it has the following density function:

$$f_x = \begin{cases} e^{-(x-\theta)} & \text{if } x \geq \theta \\ 0 & \text{if } x < \theta \end{cases}$$

θ is the unknown theoretical minimum. If $X = \{x_1, \cdots, x_n\}$ is a sample for X, then $\min_i x_i$ is a natural estimator for θ. In this situation, this estimator would be good, since its value would be very close to θ.

Case of normal populations If X follows a Gaussian distribution function $N(\mu, \sigma^2)$ ($\mu > 0$ and $\theta > 0$), then the theoretical minimum θ does not exist (because the Gaussian distribution does not have a theoretical minimum). If you consider a sample $\mathcal{X} = \{x_1, \cdots, x_n\}$, the minimum value $\min_i x_i$ is not

Value of n	20	50	100	1000	10000
Standard deviation of $\bar{X}$	0.22	0.14	0.10	0.03	0.01
Standard deviation of $\min_i x_i$	0.53	0.47	0.43	0.35	0.30

Table 1: Monte Carlo simulation of a Gaussian distribution: we see that when n the sample size increases, the sample mean converges quickly to the theoretical mean (the variance reduces quickly). However, we observe that the sample minimum has still a high variance when n the sample size increases.

Decision of the statistical test \ Truth	H_0	H_a
H_0	$1-\alpha$	β
H_a	α	$1-\beta$

Table 2: The two risk levels for hypothesis testing in statistical and probability theory. The primary risk level ($0 < \alpha < 1$) is generally the guaranteed level of confidence, while the secondary risk level ($0 < \beta < 1$)
is not always guaranteed.

a very reliable parameter of position, since it still has a high variance when the sample size n is quite large.

To illustrate this fact, let consider a simulation of a Gaussian distribution function. We use a Monte Carlo estimation of the variance of $\bar{X}$ and $\min_i x_i$ in function of n the sample size. The number of distinct samples is $N = 10000$. The simulated Gaussian distribution has a variance equal to 1. Table 1 reports the results of our simulation, and demonstrates that the variance of the sample minimum stays high when we increase the sample size n.

From all above, we clearly see that depending on the distribution function, the sample min may be a rare observation of the program execution

B Hypothesis testing in statistical and probability theory

Statistical testing is a classical mechanism to decide between two hypothesis based on samples or observations. Here, we should notice that almost all statistical tests have proved α risk level for rejecting an hypothesis only (called the null hypothesis H_0). The probability $1-\alpha$ is the confidence level of not rejecting the null hypothesis. It is not a proved probability that the alternative hypothesis H_a is true with a confidence level $1-\alpha$.

By abuse of language, we say in practice that if a test rejects H_0 a null hypothesis with a risk level α, then we *admit* H_a the alternative hypothesis with a confidence level $1-\alpha$. But this confidence level is not a mathematical one, except in rare cases.

To have more hints on hypothesis testing, we invite the reader to study section 14.2 from [6] or section B.4 from [13]. We can for instance understand that statistical tests have in reality two kinds of risks: a primary α risk, which is the probability to reject H_0 while it is true, and a secondary β risk which is the probability to accept H_0 while H_a is true, see Table 2. So, intuitively, the confidence level, sometimes known as the *strength* or *power* of the test, could be defined as $1-\beta$. All the statistical tests we use (normality test, Fisher's F-test, Student's t-test, Kolmogorov-Smirnov's test, Wilcoxon-Mann-Whitney's test) have only proved α risks levels under some hypothesis.

To conclude, we must say that hypothesis testing in statistics does not usually confirm a hypothesis with a confidence level $1-\beta$, but in reality it rejects a hypothesis with a risk of error α. By abuse of language, if a null hypothesis H_0 is not rejected with a risk α, we say that we admit the alternative hypothesis H_a with a confidence level $1-\alpha$. This is not mathematically true, because the confidence level of accepting H_a is $1-\beta$, which cannot be computed easily.

In this document, we use the abusive term confidence level for $1-\alpha$ because it is the definition used in the `R` software to perform numerous statistical tests.

C What is a *reasonable* large sample ? Observing the central limit theorem in practice

By reading all the rigorous books of statistics and probability theory [6, 8, 10, 13, 19], you would not find the answer to the golden question *how large should be a sample ?*. Indeed, there is no general answer for this question. It depends on the distributions under study. For instance, if the distribution is assumed Gaussian, we know how to compute the minimal sample size to perform a precise Student's t-test.

Some books devoted to practice [11, 15] write a limit of 30 between small ($n \leq 30$) and large ($n > 30$) samples. However, this limit is arbitrary, and does not correspond to a general definition of large samples. The number of 30 is used since long time because when $n > 30$, the values of the Student's distribution become close to the values of normal distributions. Consequently, by considering z values instead of t values when $n > 30$, the manual statistical computation becomes easier. For instance, the test of Student uses this numerical simplification. Nowadays, computers are used everywhere, hence these old simplifications become out of date.

However, we still need to have an idea about the size of a large sample if we want a general (non parametric) practical statistical protocole that is common to all benchmarks. For the purpose of defining such arbitrary size for the Speedup-Test, we made extensive experiments during multiple months. We considered two well known benchmarks families:

1. The set of SPEC OMP2001 benchmarks with various numbers of threads (from 1 thread to 8 threads). The number of distinct applications is 36.

2. The set of SPECCPU2006 applications (CFP and CINT). The number of distinct applications is 29.

We generated binaries with using the flags `-O3 --fno-strict-aliasing`. The version of `gcc` and `gfortran` is 4.3 under linux. All the applications have been executed with the train input data on two distinct execution environments:

1. *Low overhead environment:* dynamic voltage scaling is inactive, reduced OS services, no background applications, the machine executes a unique application at a time, applications are executed back-to-back.

2. *High overhead environment:* For SPEC OMP2001 applications, they are executed on a machine with a high background workload and overhead (the used machine was executing many other applications during the experiments).

Let us check if the central limit theorem is observable in such practice, and for which sample size. Our methodology is as follows:

1. Consider n the size of a sample. n is the number of runs of a benchmark. n varies between 5 and 100.

2. Consider N the number of distinct samples. Consequently, each application is run $n \times N$ times. N vries between 10 and 500. We have put a limit $n \times N = 1000$.

3. Consider the vector $\{\bar{x_1}, \cdots, \bar{x_N}\}$, the N observations of the sample mean: $\bar{x_i}$ is the sample mean of of the i^{th} sample.

4. According to the central limit theorem, If X is continuous with finite theoretical variance, and if $\bar{X}$ denotes the sample mean of a sample of size n of the distribution of X, then $\bar{X}$ should have approximatively a Gaussian distribution when n is sufficiently large. We thus have here a sample of size N of the distribution of $\bar{X}$, which will help us decide for which n this distribution can be considered as Gaussian.

5. We do the same normality analysis for the sample median $\overline{\text{med}}(X)$, that should follow a normal distribution when n is sufficiently large too.

n	5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90	95	100
$\bar{X} \not\sim N$	19	16	13	7	6	7	6	6	8	5	3	3	4	5	3	4	3	2	3	2
$\overline{\text{med}}(X) \not\sim N$	23	13	13	10	10	9	9	4	6	5	5	5	5	5	4	4	6	2	4	7

Table 3: SPEC OMP2001 on low overhead environment: The number of applications among 36 where the sample mean and the sample median do not follow a Gaussian distribution in practice in function of the sample size (risk level $\alpha = 5\%$). These measurements have been conducted on an isolated machine with *low* background workload and overhead.

n	5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90	95	100
$\bar{X} \not\sim N$	24	17	8	11	8	6	4	6	7	6	5	9	5	5	3	3	4	5	3	5
$\overline{\text{med}}(X) \not\sim N$	17	8	7	9	1	5	4	5	4	2	4	3	2	3	4	3	1	3	3	1

Table 4: SPECCPU2006 executed on low overhead environment: The number of applications among 29 where the sample mean and the sample median do not follow a Gaussian distribution in practice in function of the sample size (risk level $\alpha = 5\%$). These measurements have been conducted on an isolated machine with *low* background workload and overhead.

n	5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90	95	100
$\bar{X} \not\sim N$	26	24	23	19	19	18	18	18	15	14	16	15	16	16	14	15	15	14	14	12
$\overline{\text{med}}(X) \not\sim N$	27	24	23	23	21	21	21	20	20	19	17	17	18	16	17	13	14	11	12	15

Table 5: SPEC OMP2001 on high overhead environment: The number of applications among 36 where the sample mean and the sample median do not follow a Gaussian distribution in practice in function of the sample size (risk level $\alpha = 5\%$).

n	5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90	95	100
$\bar{X} \not\sim N$	23	23	21	20	20	21	20	20	19	20	21	20	20	19	18	16	16	14	14	15
$\overline{\text{med}}(X) \not\sim N$	23	22	22	21	21	21	20	20	21	20	20	20	21	19	19	16	18	15	15	15

Table 6: SPECCPU2006 on high overhead environment: The number of applications among 29 where the sample mean and the sample median do not follow a Gaussian distribution in practice in function of the sample size (risk level $\alpha = 5\%$).

We used the Shapiro-Wilk normality test with a risk level $\alpha = 5\%$. For the low overhead environment, Table 3 and Table 4 illustrate the number of cases that are not of Gaussian distribution if we consider samples of size n. As can be seen, it is difficult in practice to observe the central limit theorem for all sample sizes. There may be multiple reasons for that:

- The normality check test has an intrinsic error level equal to α.
- The observations x_i from X may not be totally independent observations.
- The sample size is not sufficiently large.

However, we can see that for $n > 30$, the situation is more acceptable than for $n \leq 30$. This experimental analysis show that it is not possible to decide for a fixed value of n that distinguishes between small samples and large ones. For each program, we may have a specific distribution function for its executions times. So, in theory, we should make a specific statistical analysis for each program. Since the purpose of the speedup test is to have a common statistical protocol for all situations, we accept that the arbitrary value $n > 30$ would make a frontier between small and large samples. Other values for n may be decided for specific contexts.

Table 5 and Table 6 illustrate the results of the same experiments but conducted on a high overhead environment. As can be seen, the central limit theorem is much less observable in this chaotic context. These tables strongly suggest to always make measurements on a low workload environment.

D The Wilcoxon-Mann-Whitney test

The purpose of this paragraph is to recall the relation which exists between the Wilcoxon-Mann-Whitney test and the problem of estimating or testing the probability $P(X > Y)$ and its position with regard to $1/2$.

The location shift model has been presented in Section 4.2: it states that X equals $Y + \Delta$ in distribution for some real Δ, called the shift. We thus have $F_Y(x) = F_X(x + \Delta)$ for every x, and therefore it is easy to see that

$$\mathbb{P}[X > Y] = \int_{-\infty}^{+\infty} F_Y(x)\, dF_X(x) = \int_{-\infty}^{+\infty} F_X(x + \Delta)\, dF_X(x)$$

The null hypothesis H_0 being equivalent to $\Delta = 0$, it comes

$$\mathbb{P}_{H_0}[X > Y] \;=\; \int_{-\infty}^{+\infty} F_X(x)\, dF_X(x) \;=\; \left[\frac{(F_X(x))^2}{2}\right]_{-\infty}^{+\infty} \;=\; \frac{1}{2}.$$

where $\mathbb{P}_{H_0}[X > Y]$ denotes the probability that $X > Y$ when considering that H_0 is true. Under the alternative hypothesis, we have $\Delta > 0$ (*i.e.* X tends to be greater than Y) and, due to the fact that F_X is non-decreasing (and in fact most of the time increasing on the zone where it is neither 0 nor 1), we thus have

$$\mathbb{P}_{H_a}[X > Y] >= \int_{-\infty}^{+\infty} F_X(x)\, dF_X(x) \;=\; \frac{1}{2}.$$

In addition, it has been proved by Mann and Whitney themselves ([7]) that their test is *consistent* against any alternative hypothesis for which $\mathbb{P}[X > Y] > \frac{1}{2}$ (see also [18]) : this means that the probability of rejecting H_0 tends to 1 as n becomes larger, under any framework for which $\mathbb{P}[X > Y] > \frac{1}{2}$.

Many other statistical results exist concerning the study of this probability $\mathbb{P}[X > Y]$ (including confidence bounds), we refer to [8] for some starting references about this subject.

UFR des sciences	:	45 avenue des Etats Unis. 78035 Versailles cedex
IUT de Velizy et de Rambouillet	:	10-12 avenue de l'Europe. 78140 Vélizy.
UFR des Sciences Sociales et des Humanité	:	47 boulevard Vauban. 78047 Guyancourt cedex
Faculté de droit et de science politique	:	3, rue de la Division Leclerc. 78280 Guyancourt
IUT de Mantes en Yvelines	:	7 rue Jean Hoët - 78200 Mantes la Jolie
UFR de Médecine Paris-Ile-de-France Ouest	:	9 boulevard d'Alembert Bâtiment François Rabelais. 78280 Guyancourt
Institut des Langues et des Etudes Internationales	:	5-7, boulevard d'Alembert. 78280 Guyancourt
Institut des Sciences et Techniques des Yvelines	:	45 avenue des Etas Unis - 78035 Versailles cedex
Observatoire des Sciences de l'Univers de l'UVSQ	:	11 boulevard d'Alembert. 78280 Guyancourt

Éditeur
UVSQ - Siège : 55 avenue de Paris - 78035 Versailles Cedex (France)
http://www.uvsq.fr