RANDOM SUMMATION AND ITS APPLICATION TO THE PERFORMANCE MODELLING OF COMPUTER SYSTEM

S.L. FRENKEL

The Institute of Informatics Problems, Russian Academy of Sciences,
Vavilova 44,2, 117333, Moscow, Russia, E-mail: slf-ipiran@mtu-net.ru

Abstract. This paper presents some views on the problem of application-specific systems (ASCS) performance modelling based on statistical measurement during a program's run time. The probabilistic model considered in this paper defines a performance measure relative to a given domain of application tasks, and it may be used as a base for performance verification. This measure follows a random sum of random summands, corresponding to various runtime component execution times. We consider theoretically whether the current state-of-the-art of random sums' theory can be used as the basis of applied ASCS performance modelling. Relying on the results of the analysis, it is possible to construct a mathematical model which is based on a random sum of the program's operations execution times

Keywords: performance modeling, performance evaluation, random sums, statistical prediction.

1. INTRODUCTION

Performance analysis to predict the execution time of target programs is the basis of the effective design of various application-specific computer systems [ASCS]. Let us consider the prototyping stage (either physical or virtual) of the design process (which may follow the high-level synthesis, simulation). Usually, a designer has a hardware platform variant by this time, as well as a sufficient suite of software modules. In the design phase, the designer tries to define the performance parameters of the application software by using a trace analysis tool. Let us call as "measurement-based", any approach to the execution time prediction, which is based on the results of the tracing and the profiling of a prototype of system design. Ideally, the timing performance verification at this design stage requires a mathematical model to represent the time in terms of some system characteristics, namely, in terms of the measured times of execution of some of the runtime components (e.g., function calls, basic blocks). Various mathematical models that enable the execution time prediction in terms of these measurements results have been suggested by [Saavedra and Smith, 1989; Li, 1996; Gautama, 1998]. Some of these models are deterministic in nature, and they just try to compute the execution time in terms of the various performance "indexes" (e.g., "cycles per instruction" (CPI) or their a high-level analog) [Ferrari, 1983; Saavedra, 1996;Hennessy, 1996]. Since applications' behavior are stochastic (e.g., due to data dependency in programs), the probabilistic performance models are most appropriate for this aim [Gautama, 1998]. However, these probabilistic models are not very practical, since on the one hand, they are based on some specific program models, and, on the other hand, they are fairly time-consuming. Since, in general, the execution time is a random sum of random durations of the above mentioned runtime components, which may be random values, thereby, the number of these components are also often random, the random sums theory would be very helpful for the performance modelling. This paper considers theoretically whether the current state-of-the-art of random sums theory can be used as a basis of applied ASCS performance modelling. The rest of the paper is organized as follows. Section 2 describes some related work concerning programs' execution times prediction. Section 3 describes the structure of execution time. Section 4 describes random sums concerning the execution time definition. Section 5 describes a random sum based mathematical model in terms of programs' operation execution time. Section 6 describes an example of an application-specific system performance analysis, which is based on the model. This paper, because of limited size, does not contain detailed statistical techniques description, but it only briefly explains some ideas.

2. MEASUREMENT-BASED PERFORMANCE MODELS. RELATED WORKS

Presently the prevalent approach to ASCS performance prediction is based on CPI conception (mentioned above), and represents the time spent by a processor to complete the task [Hennessy, 1996]:

$$CPU\_time = Clock\_Cycle\_Time \sum_{i=1,n} (CPI(i) * IC(i)) \qquad (2.1)$$

where CPI(i) is the average cycles per instruction for i-th instruction class (e.g. float-pointed, arithmetic, etc.), IC(i) is the counter of i-th instruction, i=1,2,...n are the indexes of the instructions set.

More advanced deterministic ways of execution time prediction have been suggested in [Saavedra, 1996], where the machine model consists of a set of abstract

 ISBN 3-936150-25-7

operations of some particular programming language. The predicted execution time of program A on machine M is performed by using detailed measured information about the execution time of these constructs. Since the models are not probabilistic, they may not provide relevant devices to estimate the performance prediction accuracy for some target applications set. Since the goal of our work is to predict the execution time of a program on an application-specific system, we have to be able to estimate both the accuracy of the prediction and the representative features of the data sample used. It is clearly an impossibility to provide it in the framework of such deterministic conceptual model.

Many probabilistic models of execution time estimation (applicable both for sequential and concurrent systems) have been suggested recently [Li, 1996; Gautama, 1998]. These models have been developed in the framework of a "task graph" conception [Adve,1993]. A task graph is a directed acyclic graph (DAG) in which nodes represent some subtasks and arcs represent the data-dependencies among the subtasks (in terms of which the program is represented). Correspondingly, DAG-based models consider the programs as a set of tasks on a particular input, and consider the program trace in terms of a task set. In [Adve, 93] the focus of the work is on the behaviour of a parallel program for a single input data set, rather than across different input sets, that takes place in case of ASCS designing. In this case, a characteristic of execution time concerning some input domain would be more interesting. The program in this model is decomposed into computationally homogeneous subtasks, and the computational requirements for each subtask are determined. The application is assumed to consist of a set of non-overlapping code segments that are totally ordered in time. The total execution time of the application is the sum of the execution times of all its code segments.

In fact, current publications [Adve, 93; Gautama, 1998] either assume that the execution time is normally distributed, or try to estimate the distributions by computing of moments of the program execution time. However, since by using various probability models we deal with decision rules like inequalities Prob(S<t), Prob(S>t), where S is a random value, t is a real one, dealing with so-called "distribution tails", the assumptions about distributions mentioned above may lead to situation of the dramatic loss of accuracy. Secondly, the rate of convergence to the normal distribution may turn out rather slow, so the using the sum may be not suitable if we consider a trace as a sequence of several quite big tasks. Besides, these models deal, in fact, only with the deterministic sums of random items, whereas, the designers often deal with unknown numbers of sequential-performed activities that determine the total time execution, for example, as a result of a non-deterministic programs behaviour due to branching, data dependencies.

In the above mentioned approaches, the programs-and-data used for the model parameters estimation are assumed as sufficient to reflect the behaviour of all application domains of the designed system. The question is whether indeed set benchmarks are able to predict the machine performance on programs not included in the benchmark suite? For example, formula (2.1) assumes, in fact, that the CPI (i) values have to be estimated from a benchmark suite, while there is no any statistical model for its rational choice.

3. STRUCTURE OF A PROGRAM EXECUTION TIME

Let us consider that we can apply special language features to split the program into intervals and to get performance characteristics for each interval. In general, the execution time of parallel programs on multiprocessor computers is determined by the various factors, e.g., a part of parallel calculations in the total volume of calculations time, or degree of overlapping of interprocessor communications with calculations. Ultimately, execution time is the maximum of the times of the program execution on each processor.

Let us call as "Application Domain" (AD) any set of applied programs with all possible input data to be executed on the designed applied platform. So, the application domain is defined by set of application programs {AP} with defined domains of their possible input data, that is an AD is a set of pairs $\{ AP_i , ID_i\}$, i is some integer, where each of AP_i is an applied program under input data ID_i .

Let O_1. O_i ., O_r be a set of some specific items ("basic operations" or some disjoint works, functions call, basic blocks), in terms of which we represent a program behaviour as a trace in a sequential manner. Each of O_i is characterized both by the type "i" (i=1,.r) and the random time of execution X_i with a distribution function $G_i(X_i)$ (in particular, they may be measured on some system prototype). If the system can execute some operations simultaneously, thanks to either multiprocessing or availability of several executive units in one uni-processor (for example, as it is performed in Alpha 21264 [Alpha, 2000]), then we may consider corresponding individual combinations of operations that can be executed simultaneously. The time of the operations execution should be considered as some random variable. Note, that this randomness may take place both in multi-processors and single-processor systems. The cache hit/miss, pipeline stalls due to hazard, may be considered as corresponding technical factors in a single processor.

When we are able to describe a program runtime as a sequence of some activity pieces (which, in general, may be some aggregations of overlapped operations [Li, 1996]), the execution time of a program APi can be expressed as a sums of some summands, corresponding to the execution times, that is expressed as N_P

$$T_P = \sum_{i=1, N_P} (X_j) \qquad (3.1)$$

where X_j is the duration of j-th sequential actions in the trace, N_P is the trace length, which should be considered as a random variable, because the different input data suites from a given program input domain may generate different paths in the program execution.

For example, let us consider the distributed execution of any applications in a p-processors's computer system [Ivannikov et al, 2000]. The application execution may be considered as an ordered system of n processes, where each of the processes is an activity dealing with a block of the application running, thereby there is a linear order of executions over the blocks set 1,.s, where s is the numbers of the blocks. Under some certain condition about the processes-and -block interactions (e.g. it is impossible to process each block more then on one processor simultaneously, each j-th blocks is distributed to j-th processor), and synchonization conditions (the end of a block in i-th processor coincides with the start of the next block execution in (i+1) processors), the time of n concurrent processes execution is

$$T = \Sigma_{i=1,n-1} \max_{1 \le u \le p} [\Sigma_{j=1,u} t_{ij} - \Sigma_{j=1,u-1} t_{i+1j}] + \Sigma_{j=i,p} t_{nj},$$

where t_{ij} is an execution time of j-th block of the i-th process.

In practice, such times can be random ones, for example, because of some pipelining effects, the number of processes n may also be dependent of the program input data, .so we deal with the random sums.

4. RANDOM SUMS MODEL OF PERFORMANCE CHARACTERIZATION

Let us consider some possibilities to estimate execution time using the random sums properties.
We can consider the set of operation execution durations (which are the summands) as a triangular array of independent non-negative integer-valued rv's $\{\tau_{ij}\}_{1 \le j \le r,}$ defined on a probability space (Ω,F,P), where F is a sigma-algebra, generated by random variables τ_{ij}, r is a number of operation types. "The columns" j=1,2….define the operations location in the sequence (in a program trace, in fact). In general, we may consider sampling sums deals with this "triangle array" scheme of random variables

$$S_n = \Sigma_{m(j)} v_{i,j} \tau_{ij} \qquad (4.1)$$

where m(j) is a sequence of integer- random rv's such that m(j) is a stopping time with respect to F_i^j, that is $\{m(j) \le 1\} \in F_i^j$), $v_{i,j}$ are random variables taking values 0 or 1, v_{ij}, τ_{ij} are independent in each row. This sum corresponds to the scheme of random sampling from a population of real numbers without replacement, where the "population" is a set of arrays $\{\tau_{ij}\}$, each of which corresponds to a program trace. Note, that for some computer architectures, the operations duration can depend on the operation execution order because of pipelining and/or caching influence [Alpha, 2000]. This circumstance is reflected in the sum (4.1), as v_{ij}, which determines implicitly the location of the operation "i" on a place "j" in program trace considered.

So, in fact, (4.1) means that any program trace is a random sample from all possible traces, generated by input data (part of which, of course, can be understood as some "controls"). Any differences between traces are reflected in their operations composition, and rv's $v_{i,j}$ just define if an operation "i" is present on j-th place in the trace, or it is absent.

The key question of the execution time model choice is whether there exists a mathematical technique to compute such a distribution.

4.1 Random Sums Theory: an Applied View

The classical random sums theory results rely essentially on the Kolmogorov-Lindeberg assumption [Zolotorev, 1997] about summands' smallness that is, in the general case, not to be justified for the execution times of program's operations. Some new results [Rahimov,1995] concern the asymptotic behaviour of (4.1) sums distribution, where the number of array rows ("i") $\rightarrow \infty$, where the above smallness assumptions have been transformed into more weak ones; that is the summand variances are decreasing as $O(i^2)$. However, this result is also not very practical, because it is asymptotical in nature, and, in fact, it requires to estimate a distribution of very sophisticated random value.

To obtain more practical results we should include in the estimation model the knowledge about the distribution of numbers of summand Np in (3.1). For some creditable assumptions about the Poisson distribution of the number of operations, a uniform estimator for the sum distribution deviation about normal law has been obtained (The Berry-Essen inequality for Poisson random sums) [Bening and

Korolev, 2002]. However, this result is also very difficult in practice.

Since our main goal is to characterize the performance relative to some set of programs ("application domain"), and, ultimately, presently the most practical execution time prediction techniques are various regression models (parametric, non-parametric), which deal with some conditional expectations values [Iverson, 1999], studying the expected value of execution time calculation issue is, perhaps, even more important then the probability distribution.

4.2 Expected Value of Random Sums

Speaking about expected values of execution time, we, in fact, should deal with various averaging techniques, that is just what we would have dealt with using any regression techniques of execution time estimation over a benchmarks set [Iverson et al, 1999].
One of the theoretical problems concerning the expected value definition is the Law of Large Numbers (LLN) conditions for the random sums. Strictly speaking, unlike the classical case of sums with non-random summands number, the limit for the "arithmetic mean" is a random value even for the independent identically distributed (i.i.d.) summands, thereby its distribution is completely determined by the asymptotic behavior of the random number of summands [Bening and Korolev, 2002]. So, one of questions is to provide some appropriate averaging.

Formally, we can rely on fact that the above triangular array model corresponds to i.i.d summands, where the distribution is a mixture of operation type duration distributions $G_i(x)$ (Section 3) that is

$$G(x)=\mathrm{Prob}(\tau_{ij} \le x) =\sum_{i=1,r} p_j G_i(x) \qquad (4.2.1)$$

where p_j are probabilities of appearance of each of the r operations type in the program traces.

Following the Wald identity, we can express the expected value of the execution time as

$$E(T_E)= E(N_p)E(x) \qquad (4.2.2)$$

where $E(N_p)$ is the expected value of the trace length N_p over an application domain considered,

$$E(X)= \int_{\Gamma x} xG(x)dx =\sum_{i=1,r} p_i \int_{\Gamma x} xG_i(x)dx = \sum_{i=1,r} p_i E(\tau_i),$$

Γ_x is an integration domain (corresponding to x definition).

So, the expected value of the execution time can be expressed as

$$E(T_E)= E(N_p)\sum_{j=1,r} p_j E(\tau_j) \qquad (4.2.3)$$

Taking into account above remark about logical difficulties of LLN performing due to randomness of the numbers of summands, the main question deals with the $\{p_j\}$ estimation, which, in classical case (when N_p is a deterministic value) would be estimated as K_i/ N_p, where K_i is the numbers of i-th operations appearances in a representative set of traces from an application domain considered. To overcome this problem let us consider an obvious way of the execution time T_E representation in terms of the i.i.d. rv's, corresponding to the execution time structure (Section 3)

$$T_E=\Sigma_{i=1,r}\Sigma_{j=1,Ki}\tau_{ij} \qquad (4.2.4)$$

where the summands in the inner sum are i.i.d. rv's, thereby, all the numbers of i-th operation appearances K_i are mutually dependent (but they are independent of the r.v's τ_{ij}), $K_1+ ... + K_r= N_p$. The Wald identity can be expressed in this case [Khokhlov, 2003] as

$$E(T_E) =\Sigma_{i=1,r} E(\tau_i)E(K_i) \qquad (4.2.5)$$

where single index "i" is used just to stand for the fact, that all operations of "i" types have the same distribution. Dividing the right side of (4.2.3) by $E(N_p)$, we obtain

$$E(T_E)/E(N_p) = \sum_{i=1,r} (E(K_i)/E(N_p))E(\tau_i) \qquad (4.2.6)$$

Comparing (4.2.3) and (4.2.6) we can obtain the probability estimator $p_i= E(K_i)/E(N_p)$.

The fact of such averaging over the total numbers of the operations in each of traces, as well as over set of each of operations type is correlated with the above mentioned character of LLN for the random sums.

The main question is what is a set of traces (program instances), over which this averaging should be performed. The answer is in interpretation both $\{K_i\}$ set and N_p as a statistics from a representative set of serially run programs which are a rational "benchmarks" set.

5. PERFORMANCE VERIFICATION

Since our main goal is a computer system performance prediction, the question is whether we are able to predict the execution time T_E of a program P of a length N_P with a sufficient accuracy as

$$T_E \approx N_P \sum_{i=1,r} p_i E(\tau_i) \qquad (5.1)$$

We may consider this approximation as a regression of T_E on N_P that is as a conditional expected value given N_P . Correspondingly, we may represent the execution time T_P estimator as

$$T_E = N_P \sum_{i=1,r} p_i E(\tau_i) + \varepsilon \qquad (5.2)$$

where ε, is stochastic, that is represents the possible errors of time prediction [Iverson99].

But what is a set of traces (program instances), over which this averaging should fulfillment to maximize (5.1) accuracy? In the most general sense this accuracy is determined by both operation times distribution and accuracy of the (p_1 ,.. , p_r) vector estimations. It would be attractive to reduce the ε error analysis (5.2) to the rate of convergence for the Law of Large Numbers (e.g., rely on Kolmogorov inequality). However, while for the non-random sums case summands dispersions impact on convergence rate follows the Kolmogorov inequality [Feller,66], the situation for random sum is considerably sophisticated [Bening and Korolev, 2002]. Note, that following the Wald identity, dispersion of the random sum can be calculated by the formula [Bening and Korolev, 2002]

$$DT_E = EN_p Dx + DN_p (Ex)^2 \qquad (5.3)$$

where its random summands have the distribution G(x) (4.2.1).

The calculation of this mean and dispersion measure can be carried out over a statistics obtaining from a considered application domain, represented by a program's set. When we are able to estimate the mean and the variance of the execution time, we will be able to estimate the accuracy of the (5.1) representation in a standard statistical manner, calculating the T_E confidence interval size as a function of the variance [Pollard, 1977]. The problem is how to choose the programs/input data ("benchmarks"), to provide these estimations. If the times τ_j are measured very accurately, than the dispersions are defined only by the values of corresponding operations K_i that are used in a program. So, we can choose a program/input data set for the performance verification so as to provide the (p_1 ,.. , p_r) vector estimation (where p_1 ,.. , p_r are from (4.2.6)) with a suitable accuracy. To achieve this we should know the distribution of vectors, representing the frequency of operations occurring in the program's trace [Frenkel, 1998]. It is easy to see that the distribution is a multinomial. Indeed, as we do not consider any information about the structure of any selected programs, then any events corresponding to the appearance of the basic operation O_i is independent of each other O_j, $i \neq j$, and the appearance of each of them does not change the appearance probability. (In other words, the independence of these events means ignoring the program's semantics).

So, knowing the frequency distribution, we can reduce the problem of benchmarks choice to the well-known problem of providing suitable confidence ellipsoids for the vectors [Pollard,77]. We may define the size and the structure of a suite of programs, that provides a suitable closeness of the frequencies, calculated over this suite, to the probabilities, used in the above formulas. This (under the above mentioned assumptions about operation's time execution) may ensure the closeness of (5.1) execution time prediction to the true value, allowing performance verification in terms of any target program execution time estimation at the design stages, when the programs may be run only on a host machine, but the operation's duration on the target HW are known with a good accuracy.
Operations duration variability impact on prediction ability depends mostly on the times distribution. Briefly, it depends mostly on the "tails" of the distributions. If their cumulative distribution functions F(x) obey the condition $1-F(x)=\sim cx^{-\alpha}$, $0<\alpha<2$, where ~ means a limit (by $x \to \infty$) of the fraction of the functions on the left and the right is 1. If F(x)) is heavy tailed then the operations' durations values shows a very high variability.

6. AN EXAMPLE

The methodology of performance evaluation was used in our practice for both FX!32 translator (in emulation mode) [Sites, 1992] of x86 applications on the Alpha platform performance optimization and the performance optimization of a RTL ("register-transver - level) model of a designed microprocessor. The emulator corresponds to a ASCS since it has been designed given all Alpha processor both hardware issues (external cache memory, register file, etc.) and Alpha Windows NT operating system. Some x86 applications sets (for example, MS-Office) could be considered as the system application domain. We can state the problem of FX!32 performance evaluation relative to the application domain. Since we have to define a probabilistic space to operate with the model, we have to understand what the randomness of the operations means in this case. Duration distributions (measured on the Alpha platform) depend on both environmental factors (Dcache miss) and some architectural properties of the emulator (e.g., how much the emulator tables match the x86 instruction structures). Some statistics can be found in http: //www.ipi.ac.ru/~lab24/frenkel.

7. DISCUSSION AND CONCLUSION

It is well-known that random summation has just as great a role in probabilistic models applications [Zolotorev, 1997; Bening and Korolev, 2002]. However, there is less evidences of its use in such important areas of contemporary computer science as "performance evaluation". In this paper we have discussed a probabilistic and statistical models for the prediction of execution times of sequential and parallel programs in a given operational and hardware environment. These models deal mostly with the some

random sums of random summands. Therefore, the main question of the modelling is whether present-day state-of-the-art of such theory, mostly concerning the LLN and CLT issues, allows us to build a well-grounded model of execution times prediction. This is possible based on expected values of the time predicted. Relying on this analysis, suggests an approach which discharges the necessary to consider summation of enormous numbers of operation times.

The results are in contrast to the traditional approaches (like [Hennessy, 1996]) which express the program execution time in terms of average of cycles per instructions. The traditional approach does not provide any devices both to estimate the accuracy and choose a reasonable benchmarks set, like the considered model, which enables these things. In fact, we encounter the usual difference between some naive statistical approaches, based only on the average numbers and counters of some events, and strong statistical methods, based on a well-grounded probabilistic model, when we can reduce the problem of benchmarks choice to the well-known problem of providing a suitable confidence ellipsoid for the estimated probabilities vector. So, in spite of numerous unresolved questions in the random sums theory, its principal results can be used effectively for performance prediction.

ACKNOWLEDGMENTS

I am very grateful to professors of Moscow State University V.Yu. Korolev and Yu.S. Khokhlov for their comments on some mathematical issues concerning this paper.

REFERENCES

Adve V. 1993, Analyzing the Behavior and Performance of Parallel Programs, Ph.D Thesis Department, Computer SciencesUniversity of Wisconsin-Madison.

Alpha 21264 Microprocessor Hardware Reference Manual, Compaq Computer Corporation, 2000.

Benning V.E., Korolev V.Yu. 2002, "Generalized Poisson Models", VSP, Utrecht.

Feller W. 1966, An Introduction to Theory of Probability and its Applications, Willey & Sons.Inc, New York.

Ferrari D. et al 1983, Measurement and Tuning of Computer Systems, Prentice-Hall.

Frenkel S. L. 1998, Performance Measurement Methodology-and-Tool for Computer System with Migrating Applied Software, *BRICS Notes Series*, NS-98-4, Aalburg, Denmark, June, Pp.83-86.

Gautama H. 1998, A Probabilistic Approach to the Analysis of Program Execution Time Technical Report No. 1-68340-44(1998)06, Faculty of Information Technology and Systems Delft University of Technology.

Hennessy J. and Patterson D. 1996, Computer Architecture: A Quantitative Approach, Second Edition, Morgan Kaufmann Publishers Inc.

Ivannikov V. P. et al 2000, On the Minimal Time Required for Execution of Distributed Concurrent Processes in Synchronous Modes, "Programming and Computer Software vol.26, N5.

Iverson M. et al 1999, Statistical Prediction of Task Execution Times through Analytic Benchmarking for Scheduling in a Heterogeneous Environment IEEE Trans. on comp., Vol 48, N 12, Pp1374-1379.

Khokhlov Yu. S. 2003, Private Communication, Moscow State University.

Li Y. A. 1996, A probabilistic framework for estimation of execution time in heterogeneous computing systems, Ph.D Thesis, the Faculty of Purdue University.

Pollard J.H. 1977, A Handbook of Numerical and Statistical Techniques, Cambridge University.

Rahimov I 1995, Random Sums and Branching Stochastic Processes, Lecture Notes in Statistics, Springer Verlag.

Saavedra R.H. and Smith A.J. 1996, Analysis of benchmark characteristics and benchmark performance prediction, ACM Transactions on Computer Systems, vol. 14, Pp. 344-384.

Sites R L. et al 1992, "Binary Translation", *Digital Technical Journal,* Vol. 4, No. 4.

Zolotarev V.M. 1997, Modern Theory of Summation of Random Variables, VSP, Utrecht.

Sergey L.Frenkel holds M.S. degree both in Radio Communication and Applied Mathematics, and Ph.D degree in Computer Science. He is a senior researcher at the Institute of Informatics Problems Russian Academy of Sciences, and he is an associate professor in Moscow State Tech. University "MIREA". His research interests mainly include probabilistic modelling of digital/computer systems. He has written papers on a variety of topics in mathematical modeling, testability, performance evaluation, as well as one textbook.