A Probabilistic Approach to Symbolic Performance Modeling of Parallel Systems

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof.dr.ir. J.T. Fokkema,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen
op donderdag, 16 december 2004 om 10.30 uur

door
Hasyim GAUTAMA

elektrotechnisch ingenieur
geboren te Jember, Indonesië

Dit proefschrift is goedgekeurd door de promotor:
Prof.dr.ir. H.J. Sips

Toegevoegd promotor: Dr.ir. A.J.C. van Gemund

Samenstelling promotiecommissie:

Rector Magnificus,	voorzitter
Prof.dr.ir. H.J. Sips,	Technische Universiteit Delft, promotor
Dr.ir. A.J.C. van Gemund,	Technische Universiteit Delft, toegevoegd promotor
Prof.dr.ir. H.E. Bal,	Vrije Universiteit Amsterdam
Prof.dr.ir. B.R. Haverkort,	Universiteit Twente
Prof.dr.ir. M. Mandjes,	Universiteit van Amsterdam and Centrum voor Wiskunde en Informatica, Amsterdam
Prof.ir. G.L. Reijns,	Technische Universiteit Delft (emeritus)
Dr. A.D. Pimentel,	Universiteit van Amsterdam

A Probabilistic Approach to Symbolic Performance Modeling of Parallel Systems
Hasyim Gautama

ISBN 90-8559-024-8

Subject headings: performance prediction, stochastic graph, workload distribution

Printed in The Netherlands

This thesis is dedicated to

Henny
Husna
Hilya
Hanif

Acknowledgments

This thesis contains the results of my research performed at the Computer Engineering (CE) and the Software Technology (ST) research groups of the department of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, the Netherlands. Numerous people have been contributed in several ways to establish the results presented here. I would like take the opportunity to thank them.

First of all, I like to thank Dr.ir. Arjan van Gemund for giving me the opportunity to do this research. I am deeply grateful to him for being my goeroe, for his guidance and help during the project, for writing the thesis, and for his feedback and advice considering my research. The existence of a better supervisor is quite unthinkable.

It is my pleasure to express my gratitude to Prof. Henk Sips for giving me the opportunity to continue my research in his ST group. Thanks to Paulo Anita for providing the technical support with the Distributed ASCI Supercomputer (DAS) for my experiments.

Furthermore, I would also like to thank Prof. Stamatis Vassiliadis for giving me the opportunity to start my research in CE group. Special thanks to Prof. Gerard Reijns for the interesting discussions and for publishing join papers. Thanks to Bert Meijs for providing the technical support for my computer.

Moreover, I would like to thank Dr. Suyono for the discussion in the probabilistic theory. Special thanks to Caswita, Darmawijoyo, Gunarjo, Hartono and all Indonesian students in Delft that I do not mention here.

I would also like to thank my ikhwah and their family in Delft: Iskandarsyah, Deden Permana and Zulfebriansyah; in The Hague: Zayd Hikmatullah and Gunaryadi; and in Groningen Elfahmi Yaman for providing spiritual support and doing *dakwah* activities together.

Most of all I would like to thank my wife, Henny Marlina, for her love, patience, understanding, moral encouragement and taking care for our children: Husna, Hilya and Hanif during my research in the Netherlands.

Delft, November 2004
Hasyim Gautama

Contents

Summary

Performance modeling plays a significant role in predicting the effects of a particular design choice or in diagnosing the cause for some observed performance behavior. Especially for complex systems such as parallel computer, typically, an intended performance cannot be achieved without recourse to some form of predictive models.

In performance prediction of parallel programs we distinguish static and dynamic prediction approaches of which the choice represents a fundamental trade-off between the amount of information and its accuracy. Static techniques offer the advantage of producing analytical information on the performance effects of symbolic program/machine parameters without requiring costly execution or multiple simulation runs for each different parameter setting or input data set. However, their limitations in modeling dynamic program behavior may have a profound negative impact on prediction accuracy. Besides dynamic behavior due to scheduling and resource contention, another important source of dynamic behavior is the dependency of a program on the input data set. For such data-dependent programs, the execution time of the program can vary greatly across the space of input data sets.

In this thesis we present a new approach to symbolic performance modeling of parallel programs that provides information on the distribution of execution times when considering a large space of input data sets. Our approach is based on statistical moments representation of distribution. We present a low-cost algorithm that computes the moments of the program execution time based on the moments associated with sequential, conditional and parallel composition. The novelty of our analysis technique is the combination between the general validity of the analysis with moment representation and ultra low solution complexity. The accuracy of our approach is experimentally evaluated for synthetic workloads as well as many empirical workloads measured from real parallel programs. Considering its ultra-low solution complexity, our approach provides an attractive cost-performance trade-off in analytical performance modeling of data-dependent parallel programs.

Chapter 1

Introduction

Parallel computer systems are being used more and more and are becoming more important economically across a whole range of application areas in science, engineering, medicine, industry, and commerce. To meet this increasing demand major vendors produce a wide variety of parallel computer systems, ranging from a cluster of work-stations connected through a local area network to special-purpose machines consisting of a large number of identical, relatively simple, processing elements. In general, high performance computing requires high arithmetic performance, high storage capacity, and fast data access and analysis.

Parallel computer systems are deployed rather than sequential ones because they are potentially more cost-effective [111]. From a hardware perspective the ratio between the price and performance of parallel processing is low. The development of very fast computer systems, i.e., systems having a faster processor, memory, and I/O, becomes ever more difficult and expensive. With the necessary and inevitably increasing complexity of the chips the probability of production or design errors steadily grows. In terms of computation power and speed (fast execution time), only parallel computer systems can handle large-scale and/or time-constrained computing problems, such as fluid turbulence and climate modeling.

Despite the above advantages, deploying parallel computer systems is not trivial since adding more processors does not always mean speedup (the ratio between sequential and parallel program execution time). Ideally, a linear speedup would occur, but often sublinear speedup or even slow down occurs. Since today's parallel computer systems are highly complex, it is not trivial to obtain speedup. Parallel computer systems are complex because they involve a wide variety of different architectures, many of which have a limited success and short life [39]. Consequently, due to these factors parallel programming is much more complex than traditional sequential programming since there exists a variety of programming models, offering various forms of parallel composition, next to sequential and conditional compositions[1]. Because of the different models, programmers sometimes have to completely rewrite algorithms, either turning from sequential to parallel and/or from one parallel version to another parallel version. Moreover, writing parallel programs involves making many decisions, such as how to partition computation tasks and data, how to map computation tasks on the available processors, and how and when to perform

[1]The various compositions are explained in Appendix B.

processor communications.

In view of the potential performance gains, the above problems related to parallel computers are nevertheless viewed as challenges. From the perspective of the programmer it is important to predict whether the intended speedup will be achieved. If the results do not meet the expectations, a programmer needs to identify the program bottlenecks. In general, programmers need to understand program behavior to be able to obtain speedup. This cannot be achieved without recourse to some form of predictive models. Hence, performance prediction plays a significant role in predicting the effects of a particular design choice or in diagnosing the cause for some observed performance behavior. In this respect performance modeling can be seen as a way to map the design of a parallel system to an optimization problem in the mathematical domain.

1.1 Performance Prediction

Performance prediction is an approach based on a computable model, ranging from a simple expression to a complex algorithm, which is generated from a (parallel) program in conjunction with a (parallel) machine. In this thesis we consider merely performance prediction of parallel programs since from our viewpoint parallel machines can also be modeled as parallel programs. Moreover, we intend to develop a general performance prediction model for parallel programs which is not machine-specific.

There are two approaches to predicting the performance of parallel programs: static and dynamic prediction; the choice between them represents a fundamental trade-off between prediction cost and accuracy. Static techniques offer the advantage that they produce analytical (and diagnostic) information on how symbolic program/machine parameters, such as the problem size, the number of processors, and the computation/communication bandwidths, affect the performance, without requiring costly execution or multiple simulation runs for each different parameter setting or input data set. However, they can only model dynamic program behavior to a limited extent, which can make their prediction inaccurate. One source of dynamic program behavior is the non-determinism introduced by dynamically scheduling tasks onto a limited number of processors, and other forms of contention for operating system services and resources like communication links, disks, memories, etc. An important example is memory hierarchies, for which it is difficult to predict whether a cache access results in a hit or a miss. Another form of dynamic behavior comes from the dependency of a program on the input data set. Especially for data processing applications such as sorting, the execution time of the program can vary greatly across the space of input data sets, even when parameters such as the data set size, are kept constant. Clearly, a static technique that cannot model execution time *distributions* is of limited practical use, in particular when one has to predict, for example, execution time bounds on hard real-time applications.

Static techniques range from low-cost symbolic techniques to numeric analytic techniques based on timed and stochastic Petri nets [45, 67, 69, 77, 78, 86, 90], queuing networks [12, 13, 38, 57, 85, 96, 116], timed process algebras [7, 15, 35, 40, 42], task graphs [27, 30, 59, 114], and hybrids (task graphs and queuing networks) [47, 48, 105]. Apart from a subset of task graphs, these techniques involve a costly numeric process in many cases (e.g., solving a Markov chain steady state equation). Compared to numeric

techniques, where the solution time complexity can range up to exponential (state space explosion), symbolic techniques are quite simple, and hence attractive. Symbolic program performance prediction is a static technique that predicts execution time in terms of a closed-form expression that retains all program parameters of interest. Hence, it provides maximum diagnostic information about the performance behavior of a program. Another motive for using symbolic performance prediction is the *solution cost*. Since the system characteristics are parametrized, due to the regularity of the parallel programs and/or machines we can usually apply *symbolic simplification*, which typically decreases evaluation complexity by orders of magnitude.

This thesis presents a symbolic, probabilistic approach to performance modeling of parallel programs that symbolically predicts the execution time distribution of a parallel program; this distribution reflects its non-deterministic (data-dependent) behavior. Our approach minimizes the solution cost while providing a prediction accuracy that is acceptable during the first phase of parallel program design. While focusing on parallel programs, our approach naturally applies to sequential programs.

Traditionally, symbolic methods model control variables such as loop bounds and branch probability values as being deterministic. For example, consider the following program fragment:

```
for (i = 1; i <= n; i++)
    if (x[i] != 0)
        x[i] = x[i] * alpha;
```

that scales a sparse double precision vector x of length n. For the purpose of the example, let the execution time be given by the double precision multiplication time τ, ignoring other program contributions. Let p denote the truth probability of the branch condition `(x[i] != 0)`. Then it follows that the mean execution time T is predicted by

$$T = \sum_{i=1}^{n} p\tau = np\tau \tag{1.1}$$

which conveniently represents T as a symbolic performance model in terms of the symbolic program parameters n, p, and τ.

While the above approach is based on the deterministic assumptions mentioned above, in practice, program parameters are often data-dependent. This implies that in our symbolic approach the values of n, p, and τ, and therefore T should be modeled as *stochastic* parameters rather than deterministic parameters, reflecting the diverse effects of input data sets. Hence, it is more appropriate to take an approach to performance prediction as illustrated in Figure 1.1. Figure 1.1 shows the performance prediction process involving the three program parameters of interest, i.e., n, p, and τ. Reflecting the diversity of a large space of input data sets, represented by the training vector of S data sets, program parameters are represented by their probability density function (pdf) rather than, e.g., (deterministic) mean values (the horizontal axis is normalized). The statistical information on n, p, and τ is assumed to be provided through either program analysis [106], user performance annotation [108], profiling information [11, 25, 91], or a combination of these (denoted by the '*' in the figure), which treatment is beyond the scope of this thesis. The performance prediction process yields an estimate of the execution time T, which is

also represented by its pdf, which is expressed as a symbolic function g of the program parameters (or some suitable representation).

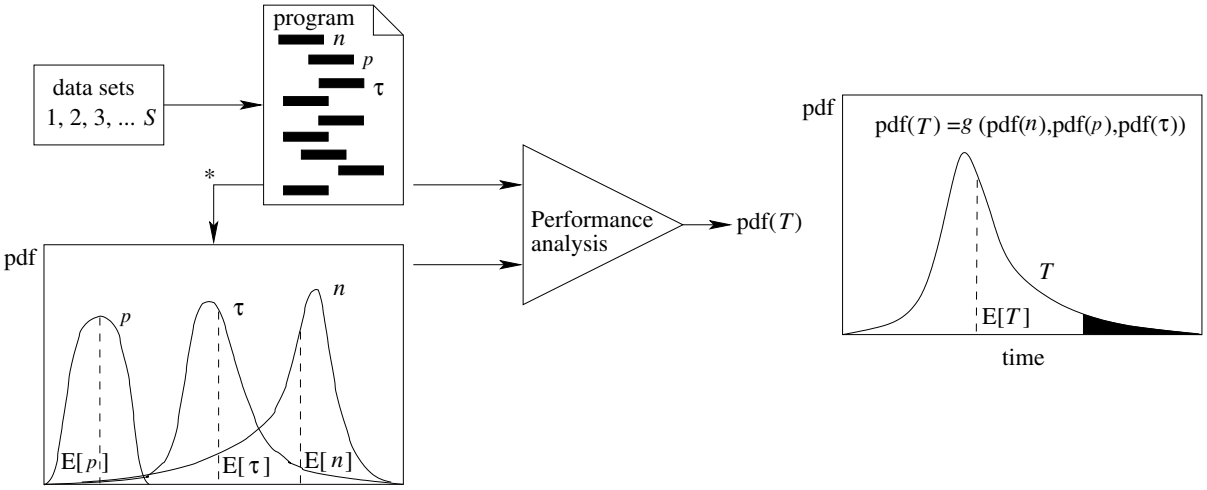

Figure 1.1: Performance prediction of data-dependent parallel programs

Figure 1.1 also illustrates that the use of deterministic mean values would only yield the mean value of T (denoted $\mathsf{E}[T]$). The mean value, however, contains limited information about T, whereas in some systems, knowledge on the (tail) distribution of program execution time may be crucial, such as in time-critical systems. Furthermore, even when only the mean execution time $\mathsf{E}[T]$ is of concern, we still require more information, such as the variance, to analyze parallel programs, than just simply the mean execution time of the tasks running in parallel. For example, consider a parallel composition of N tasks, each having a stochastic execution time X_i. The resulting execution time Y is given by

$$Y = \max_{i=1}^{N} X_i. \tag{1.2}$$

Many authors have used Eq. (1.2) as part of a static prediction technique [3, 5, 9, 18, 24, 66, 80, 92, 102, 110]. Again, in these approaches, X_i (and Y) are implicitly assumed to be deterministic. While Eq. (1.2) indeed yields a correct prediction when all X_i are deterministic, interpreting Eq. (1.2) in terms of mean values in the sense of

$$\mathsf{E}[Y] = \max_{i=1}^{N} \mathsf{E}[X_i] \tag{1.3}$$

would introduce a severe error when X_i are stochastic. This goes as follows. For example, let X_i be normally distributed with $\mathsf{E}[X_i] = 1$ and standard deviation σ. According to Eq. (1.2), $\mathsf{E}[Y]$ is given in Figure 1.2. For $\sigma = 0$, $\mathsf{E}[Y]$ is independent from N and equal to $\mathsf{E}[X_i] = 1$. For $\sigma = 1$, $\mathsf{E}[Y]$ increases logarithmically as function of N [36]. The value of $\mathsf{E}[Y]$ is approximately doubled for $N = 4$. While for $\sigma = 0$ Eqs. (1.2) and (1.3) can be applied, clearly, for $\sigma \neq 0$ applying Eq. (1.3) would result in a less accurate prediction. Thus, one cannot accurately predict the performance of (parallel) programs using only mean values. Hence, performance prediction exploiting statistical parameter information is much more effective and realistic [94].

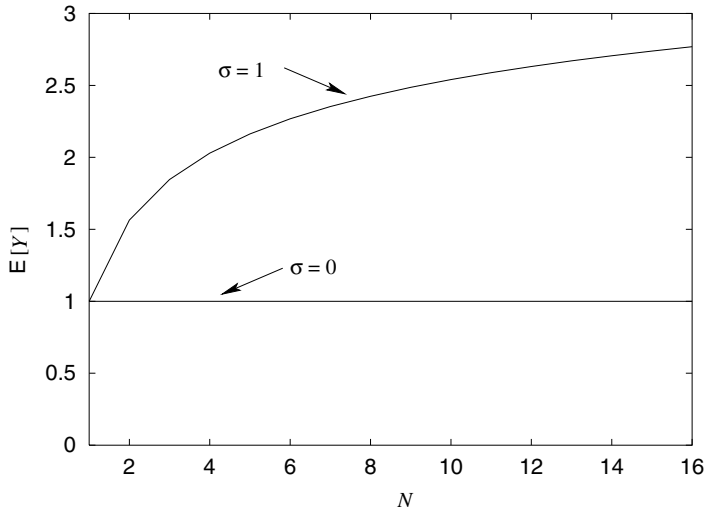

Figure 1.2: The impact of variability on parallel execution time

Ideally, a performance prediction technique will take into account all the statistical information on each program parameter, for example expressed in terms of the pdf, to avoid loss of prediction accuracy. However, such elaborate analysis is generally computationally very complex and therefore not always of practical use. In the loop program example, one would already have to compute the pdf of a product of three stochastic variables. Even if the model could be expressed symbolically, the underlying (numerical) solution procedure would be prohibitive.

1.2 Problem Statement

The choice of a distribution representation form is a determining factor in the trade-off between accuracy and solution complexity. For example, a workload representation which is based on the pdf (see, e.g., [27]), could be used to capture the statistical information, but leads to high analysis complexity. There are a number of approaches that aim at a decreasing analysis complexity by characterizing execution-time distributions through representations other than the pdf [2, 14, 59, 91, 89, 94, 103]; their parametric representations are usually based on standard distributions.

Once a workload representation has been selected, one has to determine whether it is amenable to the mathematical operations used for the different forms of control-flow, i.e., sequential, conditional, and parallel compositions. Many approaches consider stochastic basic block delays, and control variables such as loop bounds and branch probabilities [2, 29, 54, 89, 91, 109]. However, none of the above approaches provides a general method for the analysis of programs that have an *arbitrary* control structure *and* distribution, and that model data-dependent control flow in terms of stochastic variables. We will defer our discussion of related work until Chapter 2.

When using symbolic performance modeling for parallel systems based on a probabilistic approach, one will have to address the following problems:

1. To what extent can program parameters be captured in terms of statistical models?
   Considering the large space of input data sets, program variables are typically stochastic due to data dependencies in programs. Their distribution may vary across a large range of shapes.

2. How can a stochastic workload be represented symbolically?
   There is a wide range of workload representation; the choice of representation affects the trade-off between prediction accuracy and solution complexity. Most (symbolic) approaches still suffer from a high solution complexity accounting for stochastic program parameters.

3. To what extent can we assume parameters to be uncorrelated?
   Often, in a compositional analysis approach, a simplification is made by assuming program parameters to be uncorrelated to each other. In practice, such assumption is not always true.

4. To what extent can a compositional approach be applied?
   Related to the point above, one cannot always compose a model of elementary sequential, conditional, and parallel constructs while assuming that these have uncorrelated behavior.

5. What constitutes a representative training corpus?
   Since we deal with data-dependent parallel programs, the choice of a representative training corpus is significant, as parallel programs can behave significantly different on different input data sets.

We address these research questions in the context of our novel symbolic, statistical performance prediction approach, which is described in the next section.

1.3 Approach

In this thesis, we introduce a symbolic performance modeling method, where workload distribution is represented in terms of a number of *statistical moments* as a generic representation of distributions. Our motivation for using the method of moments is twofold. Our moment approach does not limit the approach to specific distributions, and is effectively a low-cost generalization of the use of mean and variance in distribution characterization as used in some of the related work. Another reason is that the method of moments is a general approach to estimate the parameters from a data set and equate the sample moments to their population counterparts.

In static techniques, a parallel program is often represented in terms of a Directed Acyclic Graph (DAG), with nodes representing tasks, and edges representing task interdependencies. In terms of this DAG, static prediction corresponds to computing the distribution of the critical path of the stochastic DAG, where each node represents a task workload distribution. For arbitrary parallel programs, the analysis of corresponding DAGs is complex since for general DAGs, the computation of the critical path involves combining path length distributions which are not always independent. In general,

only bounding approximations are possible [19, 37, 60, 98, 114], or solution techniques that are based on the assumption of an exponential workload distribution [89, 105] or based on a combination of deterministic and exponential distributions [103]. For the well-known subset of DAGs that have a recursive fork-join structure, i.e., *series-parallel* DAGs (SP-DAGs)[2], this dependency problem can be circumvented, which allows for more practical solution techniques. As many parallel algorithms can be modeled in terms of SP-DAGs [31, 34, 33], in this thesis we will only consider (recursive) fork-join programs.

In our static technique, we assume an unbounded availability of resources. Like with data-dependency, we account for the effects of scheduling and resource contention by assigning an execution time distribution to each task. While in modeling sequential and parallel control flows our approach uses DAGs, in modeling conditional control flow we extend static analysis beyond the use of DAGs. In current DAG approaches, conditional composition is *implicitly* modeled by stochastic task execution times. In our approach we model branches *explicitly* in terms of a statistical model for branch sequences.

1.4 Contributions

The contributions of this thesis are as follows.

- We develop a symbolic technique for the performance prediction of data-dependent parallel programs. Our model is based on use of moments to represent workload. Although the method of moments is not new, to our knowledge it has not been used before to model the performance of parallel computer systems.

- We present a complete analysis for parallel programs modeled in terms of sequential, conditional, and parallel compositions. The analysis produces exact solutions for the first two compositions and an approximate solution for the latter. The solution complexity is $O(1)$ for all (N-ary) compositions, while for many cases the prediction error is in the percent range.

- To validate our approach, we perform experiments using synthetic workloads as well as empirical workloads measured from real programs including the NAS Embarrassingly Parallel (NAS-EP) benchmark [8], Parallel Single-Source Shortest-Path (SSSP) [82], Parallel Sorting by Regular Sampling (PSRS) [97], WATOR [4], and Speculative Search using web search engines.

- We study to what extent our approach can deal with correlated program parameters. Since we assume that program parameters are independent, it is important to investigate the accuracy of our approach for various degrees of correlation.

- We present an implementation of our method: a symbolic performance prediction tool that fully supports the use of stochastic workloads.

[2] A DAG is said to be SP-DAG if the DAG can be reduced to a single vertex by applying series and parallel reduction rules [37]. For further explanation, we refer to Appendix B.

1.5 Thesis Outline

In the previous sections we have introduced the context of our research, and stated the challenges and contributions. The thesis is organized as follows.

In Chapter 2 we present a survey of related work on symbolic performance modeling of parallel systems. We put our approach into perspective by comparing our work to that of others, especially to approaches that use a stochastic approach to workload modeling and DAGs for control flow modeling.

In Chapter 3 we present our performance modeling methodology and the rationale behind our approach. We describe our statistical moment analysis technique and demonstrate how basic compositions of programs are analyzed. We briefly introduce the challenges related to conditional and parallel composition analysis.

In Chapter 4 we focus on the specific issues related to conditional composition. We describe the problem of branch modeling in terms of branch probability. We evaluate three statistical approaches to modeling branching behavior, viz., the Empirical approach, the Bernoulli approach, and the Alternating Renewal Process approach.

In Chapter 5 we focus on the specific issues related to parallel compositions. We describe the problem of analyzing parallel composition in terms of order statistics. We show how we solve the problem by introducing the use of Generalized Lambda Distributions as intermediate workload representation. Our symbolic solution covers and-parallel as well as or-parallel composition.

In Chapter 6 we present a tool based on our approach which is an extension of an existing tool, which only handles deterministic workload. We show that our extended tool fully supports the use of stochastic workloads. We demonstrate the tool using workloads measured from PSRS.

Finally, in Chapter 7 we summarize our work, draw the conclusions from the research, and present some recommendations for future work.

Chapter 2

Symbolic Performance Modeling

To put our moment approach into perspective, in this chapter we survey related work on symbolic performance modeling. First, we explain the principles of symbolic performance modeling. Then we characterize existing symbolic performance modeling approaches in terms of control flow modeling and the workload models used.

As mentioned in Chapter 1, symbolic performance prediction is a static technique that predicts execution time in terms of a closed-form expression that retains all program parameters of interest. Moreover, symbolic performance models offer analytic and diagnostic insight in the complex interplay of system parameters. Consider a parallel version of a simple, sparse vector scaling code shown in Figure 2.1 (left), where each processor has to scale a subvector (block partitioned). If `x[i]` is tested non-zero, then `x[i]` is multiplied by `alpha`. The corresponding symbolic performance model (based on simple, static

Program:

```
forall (p = 1; p <= P; p++)
    for (i = (p-1)*N/P+1; i <= p*N/P; i++)
        if (x[i] != 0)
            x[i] = x[i] * alpha;
```

Performance:

$$T = \max_{p=1}^{P} \sum_{i=(p-1)N/P+1}^{pN/P} \langle c_i \rangle \tau$$

Figure 2.1: A parallel version of vector scaling and the related symbolic performance model.

analysis [29]) is given in Figure 2.1 (right) where $\langle \ldots \rangle : \{true, false\} \rightarrow \{0, 1\}$ denotes Iverson's operator [43]) defined by

$$\langle c \rangle = \begin{cases} 1, & \text{if } c = true; \\ 0, & \text{otherwise,} \end{cases} \tag{2.1}$$

and τ represents the workload of the loop body (ignoring loop overhead and branch test). From the figure we can see that all system parameters (P, N, τ) are still retained in the performance model.

Symbolic performance modeling offers the potential of reducing model evaluation complexity. As parallel programs typically feature a large degree of *regularity*, the cost models are also regular, for which reason it is possible to make the evaluation cost a lot cheaper using symbolic simplification. In the above example, the regularity of the program is

shown by the fact that each processor has the same task. Furthermore, also assuming a regular vector density d (i.e., the non-zero elements are uniformly distributed across i), the summation term $\sum_{i=(p-1)N/P+1}^{pN/P} \langle c_i \rangle \tau$ reduces to $(N/P)d\tau$, which is independent from p. Consequently, T reduces to

$$T = \frac{N}{P} d\tau, \tag{2.2}$$

which has $O(1)$ evaluation complexity. This symbolic reduction is a major distinguishing feature of performance modeling using task graph representations compared to Petri nets, queuing networks, and process algebra, whose associated numerical processes are less amenable to complexity reduction. Another distinguishing feature is that the reduction process can be carried out automatically using contemporary mathematical computer-based tools.

From the above example it can be seen that the problem related to symbolic performance modeling is twofold, i.e., not only the control flow must be modeled, but also the workload. In the rest of this chapter we survey approaches in terms of these two aspects.

2.1 Control Flow Modeling

Symbolic performance modeling approaches are implicitly based on SP-DAGs, as they account for the effects of parallel and sequential task compositions in the same way: their critical path composition is isomorphic [29]. Only SP-DAGs can be represented by regular expressions and hence be treated by mathematical reduction techniques. For example, the DAG corresponding to the parallel/sequential control structure of the vector scaling code is a parallel composition of a sequential composition of tasks having a conditional workload $\langle c_i \rangle \tau$. This is isomorphic to the computation graph of T in Figure 2.1 (right). In the following, we survey SP-DAG-based approaches.

2.1.1 Deterministic DAGs

Many performance prediction approaches are implicitly based on deterministic DAGs. The reason why deterministic rather than stochastic DAGs are used is that the performance solution merely involves scalar operations. Consequently, the mathematical reduction for deterministic DAGs is much more simple than that for stochastic DAGs. Let X_i be the execution time of task i. The performance, denoted by Y, simply uses scalar addition and maximum operations as given by

$$Y = \sum_{i=1}^{N} X_i \tag{2.3}$$

and

$$Y = \max_{i=1}^{N} X_i, \tag{2.4}$$

for sequential and parallel composition, respectively.

Many deterministic DAG-based approaches have been introduced within the context of compile-time optimization since very fast predictions are required for this purpose. Approaches that fall into this category are, for example, those of Balasundaram, Fox,

Kennedy, and Kremer [9], Sarkar [92], Atapattu and Gannon [5], and Van Gemund [29]. The approach of Van Gemund is important in the context of this thesis since we present its implementation in Chapter 6 and compare it with our extended version, which fully supports the use of stochastic workloads.

2.1.2 Stochastic DAGs

As mentioned earlier, a fundamental limitation of deterministic DAGs is that they cannot model the data variability introduced by non-deterministic task completion times. In order to model data dependency in parallel programs, stochastic values are used to model the task workloads in the DAGs. Unlike deterministic DAGs, the complexity of critical path analysis is prohibitive unless some restrictions are introduced: either the scope of graph structure must be limited or the presentation of workload must be limited to specific distributions. Similar to our symbolic approach, in most work only SP-DAGs are considered because of the analytical problems that arise when non SP-DAGs are used, as mentioned in Chapter 1. For the SP-DAGs case, the performance evaluation of sequential and parallel compositions merely becomes a matter of convolution and order statistics, respectively. Consider the example corresponding to Eqs. (2.3) and (2.4). Now, let X_i be stochastic and mutually independent. For sequential composition, the pdf of Y is given by the convolution of the individual pdf's according to

$$f_Y(y) = f_{X_1}(x) \star f_{X_2}(x) \star \ldots \star f_{X_N}(x), \tag{2.5}$$

where

$$f_{X_i}(x) \star f_{X_j}(x) = \int_{-\infty}^{\infty} f_{X_i}(x) f_{X_j}(y-x) \mathrm{d}x. \tag{2.6}$$

For parallel composition, the cumulative distribution function (cdf) of Y is the product of the individual cdfs [106] according to

$$F_Y(y) = \prod_{i=1}^{N} F_{X_i}(y). \tag{2.7}$$

Unlike that of Eqs. (2.3) and (2.4), the solution complexity of Eqs. (2.6) and (2.7) is generally very high. We describe these issues further in Chapter 3.

2.1.3 Branch Modeling

Unlike sequential and parallel compositions, conditional composition in terms of branching cannot be modeled explicitly by DAGs. To the best of our knowledge there exists no symbolic performance modeling work which also deals with branching, except the work of Sarkar [91] and Van Gemund [32]. Their work is based on the use of a single, deterministic parameter to model branching probability. In contrast, in this thesis, we extend this approach by modeling branch behavior in a much more general statistical manner. This approach is further elaborated in Chapter 4.

2.2 Workload Modeling

As mentioned in Chapter 1, the trade-off between prediction accuracy and solution complexity is largely determined by how the execution time distributions are represented. We argued in Section 1.1 that one needs more than a deterministic model to predict the execution time of data-dependent parallel programs. In this section, we review approaches which use stochastic models for the workload of programs, e.g., the execution time of basic blocks and program parameters such as loop bounds. In terms of Figure 2.1, workload modeling is related to modeling the execution time of τ, d and N.

To avoid a complicated analysis of sequential and parallel compositions, most approaches based on stochastic models assume parameters to be independent. For sequential composition most of the related work aims to predict the mean and variance of Eq. (2.5). Let $\mathsf{E}[Y]$ and $\mathsf{Var}[Y]$ denote the mean and variance of Y, respectively. Then the mean and variance of Y in Eq. (2.5) are given by

$$\mathsf{E}[Y] = \sum_{i=1}^{N} \mathsf{E}[X_i] \quad \text{and} \tag{2.8a}$$

$$\mathsf{Var}[Y] = \sum_{i=1}^{N} \mathsf{Var}[X_i], \tag{2.8b}$$

respectively. In contrast to sequential composition, parallel composition poses more analytical problems. In the following, we review approaches based on stochastic models which consider binary and/or N-ary parallel composition. We assume that parameters are independent.

2.2.1 Pdf-based Models

An approach using the pdf has been described by Gelenbe to determine the completion times of SP-DAGs [27]. However, the high-cost numerical integration has a serious drawback regarding practical use since the cost of computing integrals, e.g., Eq. (2.6), increases linearly as a function of the considered time domain.

Another way of characterizing the pdf is based on series approximation, for example the Gram-Charlier series of type A. We show in Appendix D that the analysis is asymptotically exact. Unfortunately, the number of Gram-Charlier terms needed for a sufficiently accurate approximation is prohibitive.

Lester approximates the pdf using the z-transform [59]. For example, if X is normally distributed with parameter μ and σ, the probability mass function (pmf) of X can be approximated by

$$f_X(x) = .00621(x^{\mu-3\sigma}+x^{\mu-3\sigma})+.0606(x^{\mu-2\sigma}+x^{\mu-2\sigma})+.2417(x^{\mu-\sigma}+x^{\mu-2\sigma})+.3829x^{\mu}. \tag{2.9}$$

In particular, sequential and conditional compositions can be easily expressed in terms of the z-transform. While the real pdf can be approximated well, the solution complexity of the underlying numerical process is still high.

Schopf and Berman [93] use histograms with a limited number of intervals. However, the analysis complexity grows rapidly with the number of histogram intervals needed

to accurately characterize a distribution. Also Lüthi *et al.* [62] characterize parameter variabilities in terms of histograms. In contrast to program (task graph) analysis, they address the problem of solving queuing models with load variabilities.

2.2.2 Specific Distribution Models

Many approaches reduce solution complexity by using specific distributions that are characterized by a limited number of parameters: generality is traded for cost reduction. Thomasian and Bay [105] consider the exponential distribution. For example, let X be exponentially distributed with parameter θ, then the rth moment of Y in Eq. (2.7) for independent and identically distributed (iid) X_i is given by [106]

$$\mathsf{E}[Y^r] = \sum_{i=1}^{N} \binom{N}{i} (-1)^{i-1} \frac{r!}{(i\theta)^r}. \tag{2.10}$$

Since program parameters are typically correlated, exponentially distributed program workloads are hardly found in practice.

Sötz uses an exponential distribution with parameter θ combined with a deterministic offset d. His approximation is based on the use of Erlang distributions with mean μ and standard deviation σ. Then the parameters can be found by $\theta = 1/\sigma$ and $d = m - \sigma$. While the analysis is straightforward, the approach may introduce significant errors [103].

Mak and Lundstrom [64] use Erlang distributions[1] instead of using exponentially distributed task times. The Erlang distribution corresponds to a series of r identical exponentially distributed stages each with a mean of $1/\lambda$. Using Erlang distributions, they analyze Eq. (2.7) for $N = 2$ (binary parallel composition) as follows. Let $X_1(\lambda_1, r_1)$ and $X_2(\lambda_2, r_2)$ be Erlang random variables. For binary parallel composition it holds that

$$\begin{aligned}\mathsf{E}[Y] = \mathsf{E}[X_1] + \mathsf{E}[X_2] &- \frac{\lambda_1^{r_1}}{(\lambda_1+\lambda_2)^{r_1+1}} \sum_{k=0}^{r_2-1} \left(\frac{\lambda_2}{\lambda_1+\lambda_2}\right)^k \frac{(r_1+k)!}{(r_1-1)!k!} \\ &- \frac{\lambda_2^{r_2}}{(\lambda_1+\lambda_2)^{r_2+1}} \sum_{k=0}^{r_1-1} \left(\frac{\lambda_1}{\lambda_1+\lambda_2}\right)^k \frac{(r_2+k)!}{(r_2-1)!k!},\end{aligned} \tag{2.11a}$$

$$\begin{aligned}\mathsf{Var}[Y] = \mathsf{E}[X_1]^2 + \mathsf{E}[X_2]^2 + \mathsf{Var}[X_1] + \mathsf{Var}[X_2] - \mathsf{E}[Y]^2 \\ - \frac{\lambda_1^{r_1}}{(\lambda_1+\lambda_2)^{r_1+2}} \sum_{k=0}^{r_2-1} \left(\frac{\lambda_2}{\lambda_1+\lambda_2}\right)^k \frac{(r_1+k+1)!}{(r_1-1)!k!} \\ - \frac{\lambda_2^{r_2}}{(\lambda_1+\lambda_2)^{r_2+2}} \sum_{k=0}^{r_1-1} \left(\frac{\lambda_1}{\lambda_1+\lambda_2}\right)^k \frac{(r_2+k+1)!}{(r_2-1)!k!}.\end{aligned} \tag{2.11b}$$

Liang and Tripathi [61] derive the mean and variance of parallel composition for Erlang and/or hyperexponential distributions[2]. The use of both distributions depends on the coefficient of variation. Let $X_1(\lambda_{11}, \lambda_{12}, l_1,)$ and $X_2(\lambda_{21}, \lambda_{22}, l_2)$ be hyperexponential random

[1] The cdf of Erlang random variable X is given by $F_X(x) = 1 - \sum_{k=0}^{r-1} \frac{(\lambda x)^k}{k!} e^{-\lambda x}$, where $x > 0$ and $\lambda > 0$.

[2] The cdf of hyperexponential random variable X used in [61] is given by $F_X(x) = l(1 - e^{-\lambda_1 x}) + l(1 - e^{-\lambda_2 x})$, where $x > 0, \lambda_1, \lambda_2 > 0$, and $0 \leq l \leq 1$.

variables. For binary parallel composition it holds that

$$\begin{aligned}\mathsf{E}[Y] &= \mathsf{E}[X_1] + \mathsf{E}[X_2] \\ &\quad - \frac{l_1 l_2}{\lambda_{11}+\lambda_{21}} - \frac{(1-l_1)l_2}{\lambda_{12}+\lambda_{21}} - \frac{l_1(1-l_2)}{\lambda_{11}+\lambda_{22}} - \frac{(1-l_1)(1-l_2)}{\lambda_{12}+\lambda_{22}}, \qquad (2.12a)\\ \mathsf{Var}[Y] &= \mathsf{E}[X_1] + \mathsf{E}[X_2] + \mathsf{Var}[X_1] + \mathsf{Var}[X_2] - \mathsf{E}[Y]^2 \\ &\quad - \frac{2l_1 l_2}{\lambda_{11}+\lambda_{21}} - \frac{2(1-l_1)l_2}{\lambda_{12}+\lambda_{21}} - \frac{2l_1(1-l_2)}{\lambda_{11}+\lambda_{22}} - \frac{2(1-l_1)(1-l_2)}{\lambda_{12}+\lambda_{22}}. \qquad (2.12b)\end{aligned}$$

If the distribution of X_i are mixed, the mean and variance of Y are as follows. Let $X_1(\lambda_{11}, \lambda_{12}, l_1,)$ and $X_2(\lambda_2, r_2)$ be hyperexponential and Erlang random variables, respectively. For binary parallel composition it holds that

$$\begin{aligned}\mathsf{E}[Y] &= \mathsf{E}[X_2] + \frac{l_1}{\lambda_{11}}\left(\frac{\lambda_2}{(\lambda_{11}+\lambda_2)}\right)^{r_2} + \frac{1-l_1}{\lambda_{12}}\left(\frac{\lambda_2}{(\lambda_{12}+\lambda_2)}\right)^{r_2}, \qquad (2.13a)\\ \mathsf{Var}[Y] &= \mathsf{E}[X_2]^2 + \mathsf{Var}[X_2]^2 - \mathsf{E}[Y]^2 \\ &\quad + 2\left[\frac{l_1}{\lambda_{11}^2}\left(\frac{\lambda_2}{(\lambda_{12}+\lambda_2)}\right)^{r_2} + \frac{(1-l_1)}{\lambda_{12}}\left(\frac{\lambda_2}{(\lambda_{12}+\lambda_2)}\right)^{r_2}\right] \\ &\quad + 2\frac{r_2}{\lambda_2}\left[\frac{l_1}{\lambda_{11}^2}\left(\frac{\lambda_2}{(\lambda_{11}+\lambda_2)}\right)^{r_2+1} + \frac{(1-l_1)}{\lambda_{12}}\left(\frac{\lambda_2}{(\lambda_{12}+\lambda_2)}\right)^{r_2+1}\right]. \qquad (2.13b)\end{aligned}$$

Sahner and Trivedi [89] use exponomial distributions[3]. Exponomial distributions include exponential, hyperexponential and Erlang distributions, and mixtures of Erlang distributions. Their technique uses the fact that exponomial distributions are closed under various operations including maximum. Let X_1 and X_2 be exponomial random variables having cdfs $F_{X_1}(x) = 1 - e^{-4x}$ and $F_{X_2}2(x) = 1 - e^{-5x}$, respectively. For binary parallel composition, it holds that

$$\begin{aligned}F_Y(x) &= F_{X_1}(x)F_{X_2}(x) \\ &= (1-e^{-4x})(1-e^{-5x}) \\ &= 1 - e^{-4x} - e^{-5x} + e^{-9x}, \qquad (2.14)\end{aligned}$$

where $F_Y(x)$ is also an exponomial distribution. While exponential, Erlang, hyperexponential, and exponomial workloads offer low cost, analytic tractability, and are appropriate for, e.g., reliability modeling, such workloads are often only a coarse approximation of the workloads as measured in real programs. Note that these distributions are all special cases of the phase-type distribution [71] which is characterized by a Markov chain and a transition probability matrix. Although the phase-type distribution are computationally efficient, but the distribution applies only to restrictive models when the fitting method is based on the statistical moments [46].

Schopf and Berman [95] use normal distributions. While the application to sequential programs is straightforward, binary parallelism is approximated heuristically, entailing large errors when both workloads are similar (see Section 5.3.1).

[3] A random variable X is said to be exponomially distributed if the cdf can be expressed as $F_X(x) = \sum_i a_i x^{k_i} e^{b_i x}$.

2.2.3 Arbitrary Distribution Models

Having the purpose to extend the analysis to more arbitrary workloads, other approaches approximate the workload distribution in terms of, e.g., mean, variance, and/or bounds. Gumbel [36] approximates the mean execution time of a parallel composition with N tasks having iid symmetric distributions, provided that their mean ($\mathsf{E}[X_i]$)and variance ($\mathsf{Var}[X_i]$) are known. The approximation is given by

$$\mathsf{E}[Y] \approx \mathsf{E}[X_i] + \sqrt{2\mathsf{Var}[X_i]\log(0.4N)}. \tag{2.15}$$

Axelrod [6] also uses Eq. (2.15) to approximate the execution time of parallel tasks with synchronization barriers on multiprocessors. Under the same assumptions, Robinson [87] introduces upper and lower bounds on the mean execution time while allowing dependencies among subtasks. These bounds were later improved by Madala and Sinclair [63]. To include wider-than-symmetric distributions, Kruskal and Weiss [54] use increasing failure rate (IFR) distributions[4] to approximate the mean execution time of parallel compositions for iid subtasks. IFR includes exponential, gamma with $\mu/\sigma \geq 1$, Weibull with rate ≥ 1, truncated normal, i.e., normal distribution constrained to be positive, and uniform on the interval $(0, c)$ for any $c > 0$.

While the approximation error of these approaches is quite reasonable, only the first moment can be obtained. As, in turn, mean and variance are required inputs, this approach cannot be applied to DAGs with nested parallelism, thus seriously limiting their applicability.

2.2.4 Moment-based Models

There are other approaches that also characterize the execution time distribution in terms of the mean and variance. These approaches include the work of Sarkar [91] for sequential compositions. Although they are not aimed at analyzing parallel composition, they show that moment-based models are straightforward for the analysis of sequential compositions.

Reijns *et al.* [84] use Pearson distributions for the analysis of parallel composition. Although Pearson distributions include a wide range of distributions, the associated analysis does not always yield a closed-form expression, e.g., for a parallel composition with N tasks having iid, normally distributed workloads.

2.3 Summary

In this chapter we have reviewed approaches that propose the use of cdf or pdf, and other representations that impose restrictions on the allowable distributions. Some choose a characterization in terms of mean and variance, which allows arbitrary distributions while

[4] A random variable X is said to be IFR if the following equation holds

$$F_X(x) = \begin{cases} 0, & \text{if } x = 0, \text{ positive random variable,} \\ \dfrac{1 - F_X(x+\epsilon)}{1 - F_X(x)}, & \text{if } \epsilon > 0, \text{ monotone decreasing in } x. \end{cases} \tag{2.16}$$

still sacrificing accuracy. None of the approaches addresses the effect of stochastic loop bounds or branching, which are essential in stochastic program modeling. An exception is the approach taken by Adve and Vernon [1] which allows a sequential loop bound to be stochastic.

Table 2.1 summarizes the related work in symbolic performance modeling in terms of the distribution type used, and whether the approach addresses sequential composition (SC), stochastic loop bounds (LB), condition probabilities (CP), binary parallel composition (BP) and N-ary parallel composition (NP). NA means that the approach is not applicable to the corresponding composition. Our approach is included for reference.

Table 2.1: Summary of related work in symbolic performance modeling.

First author	Distribution type	SC	LB	CP	BP	NP
Adve [1]	Mean & Var.	NA	√	NA	NA	NA
Axelrod [6]	Normal	NA	NA	NA	NA	√
Gautama (Appendix D)	Series	NA	NA	NA	√	√
Gelenbe [27]	Pdf	√	NA	NA	√	√
Gumbel [36]	Normal	NA	NA	NA	NA	√
Kruskal [54]	IFR	NA	NA	NA	NA	√
Lester [59]	Z-transform	√	NA	NA	NA	√
Liang [61]	Erlang & Hyperexp.	√	NA	NA	√	√
Lüthi [62]	Histogram	√	NA	NA	√	NA
Madala [63]	Normal	NA	NA	NA	NA	√
Mak [64]	Erlang	√	NA	NA	√	√
Reijns [84]	Pearson	NA	NA	NA	√	√
Robinson [87]	Normal	NA	NA	NA	NA	√
Sahner [89]	Exponomial	√	NA	NA	√	√
Sarkar [91]	Mean & Var.	√	NA	NA	NA	NA
Schopf [93]	Histogram	√	NA	NA	√	NA
Schopf [95]	Normal	√	NA	NA	√	NA
Sötz [103]	Det & Exp	√	NA	NA	√	√
Gautama (this thesis)	Moments (4)	√	√	√	√	√

Chapter 3

Statistical Moment Analysis

As mentioned in Section 1.3, our performance prediction approach is based on modeling workload in terms of a number of statistical moments, in order to combine good accuracy with minimum solution complexity. In this chapter we introduce our statistical moment approach. First, we present the rationale, modeling methodology, and the analysis of sequential and conditional compositions. Second, we formulate the parallel composition problem. We then proceed with the analysis of sequential, conditional, and parallel composition in terms of our moment approach.

3.1 Rationale

The choice for using statistical moments to characterize the pdf is primarily based on two reasons. First, our moment approach is effectively a generalization of the use of mean and variance in distribution characterization. Like the use of mean and variance, the associated benefit is a low analysis complexity. Unlike the mean and variance approach, however, our approach captures essential information on the cdf (e.g., upper percentiles, which are relevant for time-critical applications), while retaining the low-complexity benefit. Second, the method of moments is a general approach to estimate the parameters from a data set and to equate the sample moments to their population counterparts. Another approach such as Maximum Likelihood is unsuited for many practical purposes because it does not reveal the origin of the actual value of the parameters [104]. Although in general, the method of moments does not completely determine a cdf, in our case, knowledge of all moments is equivalent to knowledge of the distribution since execution time distributions can be assumed to be finite [104]. For detailed discussion of the method of moments, we refer to Appendix A.

Theoretically, an unlimited number of statistical moments can be incorporated in our performance prediction approach. However, we only consider the first four moments for the following three reasons. First, lower moments are more important than higher moments for the characterization of a distribution, as has been frequently discussed in statistics, such as for Pearson distributions and Johnson distributions. It has been shown [83] that the first four moments allow to reconstruct and approximate the original distribution while introducing an acceptable error, and to distinguish between well-known standard distributions. Second, in measurements, lower moments are more robust than higher mo-

ments. The measured values of higher moments are so sensitive to sampling fluctuations that including higher moments in the analysis will not always improve the prediction accuracy. Finally, our analysis of parallel composition is specifically aimed at using only the first four moments (see Chapter 5).

3.2 Methodology

Our approach towards program performance prediction is depicted in Figure 3.1. In the figure (left), we distinguish between modeling approach (top) and the measurements (bottom). In the figure (right), we show the focus of our research, i.e., the moment

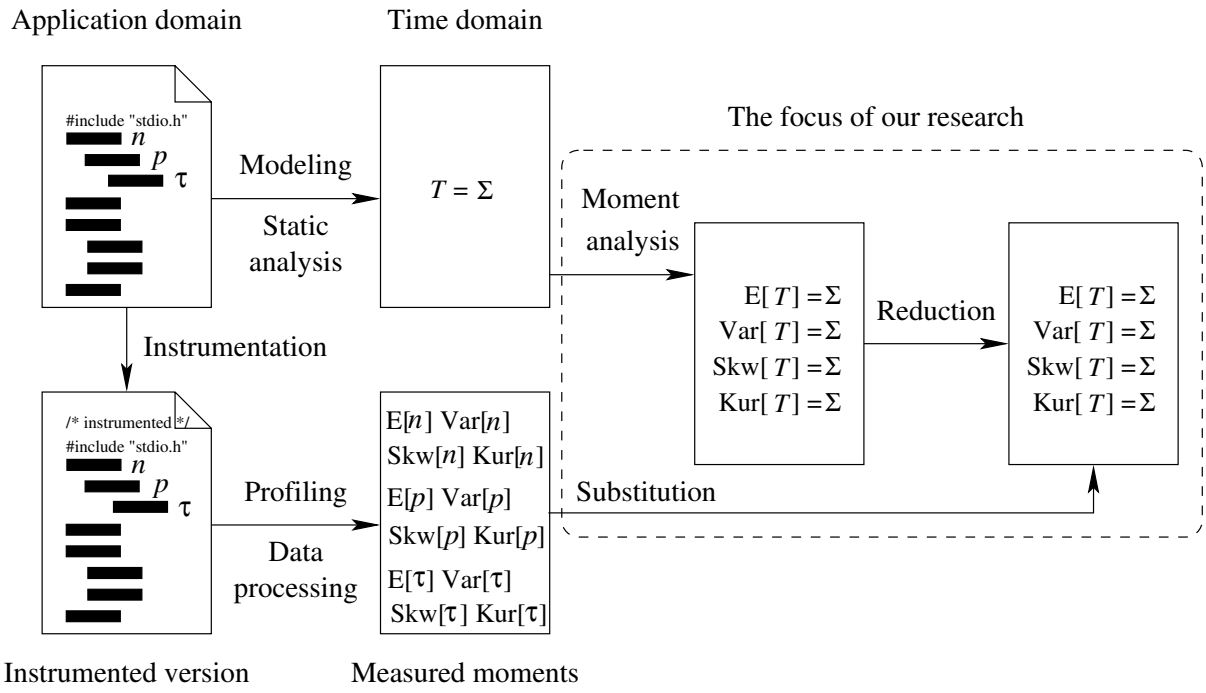

Figure 3.1: Our performance prediction approach

analysis. As described in Chapter 2, due to program regularity (sequential and parallel composition represented by SP-DAGs), the moment expression can be reduced. To verify the prediction accuracy, we will compare the predicted execution time with the measured one.

3.2.1 Modeling Approach

In the first step of our modeling approach, the source code is modeled (Figure 3.1, top left). This modeling step yields a performance simulation model in which all data dependencies have been removed, isolating only those terms that are relevant for performance modeling. Because the program is in SP form, in the modeling step we can use a performance modeling language, which is reminiscent to ordinary mathematics in the sense

that deriving formulae for the execution time is straightforward. Subsequently, we apply static analysis, which yields the program execution time.

3.2.2 Measurements

As mentioned in Section 2, the purpose of the measurements we perform is to evaluate our analytical prediction. Since the focus of this thesis lies in developing an analysis technique, we have kept the applied measurement technique simple.

Profiling

The first step of our measurement method is instrumenting the source code with counters, known as *counter-based* profiling. This profiling method probes the frequency of control constructs rather than counting the execution frequency of each basic block. If the instrumented program terminates, output files containing all counters are saved. We run a program under study many times to obtain data samples. We also instrument the source code with a global counter to measure the program execution time. The use of counter-based profiling is attractive since this way of profiling is machine-independent.

In the following, we show how the program described in Section 1.1 is instrumented with counters. For a much longer instrumented program example we refer to Appendix E.

```
for_call_c++;
for (i = 1; i <= n; i++) {
    for_true_c++;
    if_call+c++;
    if (x[i] != 0) {
        if_true_c++;
        x[i] = x[i] * alpha;
        exec_time_c++;
    }
}
```

All counter values are initialized to zero. Each `if`, `for`, or `while` control construct always has two counters. The first counter, `_call_c`, is placed right before the control construct, and the second counter, `_true_c`, is placed right after the control construct. If the control construct is executed, then `_call_c` is incremented. And if the condition is true, that is, if the statement inside the control construct is executed, `_true_c` is incremented. The frequency of a control construct is obtained by dividing `_true_c` by `_call_c`.

Data Processing

After instrumentation, the program is ready to be compiled and executed. Instrumented source codes take a bit longer to execute than the original source code. Obviously, additional time is needed to increment the counters and to execute input and output file functions. The measurement is repeated for different sample data sets to obtain the data samples.

After the data samples are obtained, the samples are processed to obtain the first four moments of measured variables. The results of the data processing are moments of control constructs as well as the total execution time T, which is used to verify the predicted execution time.

3.3 Preliminaries

This section defines the terminology of probability theory (mostly taken from [58, 106]) which we use in this thesis. Readers who are familiar with the terminology can skip this section. We begin by discussing the notion of random experiments, conditional probability, random variables, the cumulative distribution function, and the probability density function. Based on the continuity of the cdf, we distinguish two types of random variables, i.e., discrete and continuous random variables.

Since statistical moments of random variables are an important issue in this thesis, we introduce the notion of expected values and moments. We present the relation between the raw and central moments of random variables. It is discussed what individual effect the first four central moments have on the distribution of random variables. Furthermore, transform methods are introduced, which are used in our analysis, in which two generating functions are presented, i.e., the moment generating function (mgf) and the probability generating function (pgf). Statistical operators are printed in the sans serif font, while abbreviations are printed in normal fonts.

3.3.1 Probability and Random Variables

A *random experiment* is an experiment in which the outcome varies in an unpredictable fashion when the experiment is repeated under the same conditions [58]. An *outcome* of a random experiment is defined as a result that cannot be decomposed into other results, while the set of all possible outcomes is defined as the *sample space* (S). A subset of S is defined as an *event*. Given a sample space S, a *probability measure* P on S is a rule that assigns to each event E a real number $\mathsf{P}[E]$.

The *conditional probability* is defined by

$$\mathsf{P}[E_1|E_2] = \frac{\mathsf{P}[E_1 \cap E_2]}{\mathsf{P}[E_2]} \qquad \text{for } \mathsf{P}[E_2] > 0. \tag{3.1}$$

We define two events E_1 and E_2 to be *independent* if

$$\mathsf{P}[E_1 \cap E_2] = \mathsf{P}[E_1]\mathsf{P}[E_2]. \tag{3.2}$$

We define *cumulative distribution function* (cdf) X according to

$$F_X(x) = \mathsf{P}[X \leq x] \qquad \text{for } -\infty < x < +\infty. \tag{3.3}$$

Based on the continuity of the cdf, we define two types of random variables [58]. A *discrete random variable* is defined as a random variable whose cdf is a right-continuous staircase function of x, with jumps at a countable set of points $x_0, x_1, x_2, \ldots$. Discrete random variables take on values from a finite or at most a countably infinite set $S_X = \{x_0, x_1, x_2, \ldots\}$.

They arise mostly in applications that usually have $S_X = \{0, 1, 2, \ldots\}$. The cdf of a discrete random variable can be written as the weighted sum of unit step functions

$$F_X(x) = \sum_i p_X(x_i) u(x - x_i), \tag{3.4}$$

where $p_X(x_i) = P[X = x_i]$ gives the magnitude of the jumps in the cdf. The set of probabilities $p_X(x_i) = P[X = x_i]$ of the elements in S_X is known as the *probability mass function* (pmf) of X.

A *continuous random variable* is defined as a random variable whose cdf $F_X(x)$ is continuous everywhere, and which, in addition, is sufficiently smooth, so that it can be written as an integral of some non-negative function $f(x)$:

$$F_X(x) = \int_{-\infty}^{x} f(t)\, \mathrm{d}t. \tag{3.5}$$

The *probability density function* (pdf) of X is defined as the derivative of $F_X(x)$:

$$f_X(x) = \frac{\mathrm{d}F_X(x)}{\mathrm{d}x}. \tag{3.6}$$

3.3.2 Expected Values and Moments

The *expected value* or *mean* of a random variable X, denoted by $\mathsf{E}[X]$, is defined by

$$\mathsf{E}[X] = \begin{cases} \sum_i x_i p_X(x_i) & \text{if } X \text{ is discrete,} \\ \int_{-\infty}^{\infty} x \, \mathrm{d}F_X(x) & \text{if } X \text{ is continuous.} \end{cases} \tag{3.7}$$

The first expression is obtained by substituting Eq. (3.4) into Eq. (3.7), while the second expression is valid provided that the integral exists. Eq. (3.7) also defines the expectation of any function of X, say $h(X)$. Since $h(X)$ is itself a random variable, it follows from Eq. (3.7) that

$$\mathsf{E}[h(X)] = \int_{-\infty}^{\infty} x \, \mathrm{d}F_{h(X)}(x), \tag{3.8}$$

where $F_{h(X)}$ is the cdf of $h(X)$. If $h(X) = X^r$, the expected value of the function $h(X)$ is called the rth *raw moment* of random variable X according to

$$\mu'_r = \mathsf{E}[X^r] = \int_{-\infty}^{\infty} x^r \, \mathrm{d}F(x), \quad \text{for } r = 1, 2, 3, \ldots. \tag{3.9}$$

The rth central moment, μ_r, is defined as follows

$$\mu_r = \mathsf{E}[(X - \mathsf{E}[X])^r] = \int_{-\infty}^{\infty} (x - \mathsf{E}[X])^r \, \mathrm{d}F(x). \tag{3.10}$$

Specifically, μ_2 is called the *variance* of X, also denoted by $\mathsf{Var}[X]$, while the positive square root of the variance is called the standard deviation, and denoted by σ or $\mathsf{Std}[X]$ according to

$$\sigma = \mathsf{Std}[X] = |\sqrt{\mu_2}|. \tag{3.11}$$

The dimensionless ratio of μ_3 to σ^3 is called the *skewness* of the distribution, denoted by $\mathsf{Skw}[X]$, and the ratio of μ_4 to σ^4 is called the *kurtosis* of the distribution, denoted by $\mathsf{Kur}[X]$. That is,

$$\mathsf{Skw}[X] = \frac{\mu_3}{\sigma^3}, \tag{3.12}$$

$$\mathsf{Kur}[X] = \frac{\mu_4}{\sigma^4}. \tag{3.13}$$

The graphical interpretation of the first four moments is shown in Figures 3.2 and 3.3, respectively. The mean represents the center of mass of the distribution as depicted in Figure 3.2 (left). This figure also shows the notion of the *mode* of distribution that is the value of random variable at which the pdf or pmf peaks. The variance is a measure of dispersion of the random variable around $\mathsf{E}[X]$. The smaller the variance, the more sharply the pdf is concentrated around $\mathsf{E}[X]$ as shown in Figure 3.2 (right). The skewness is a measure of asymmetry of the distribution while the kurtosis represents the degree of peakedness of the distribution as shown in Figure 3.3.

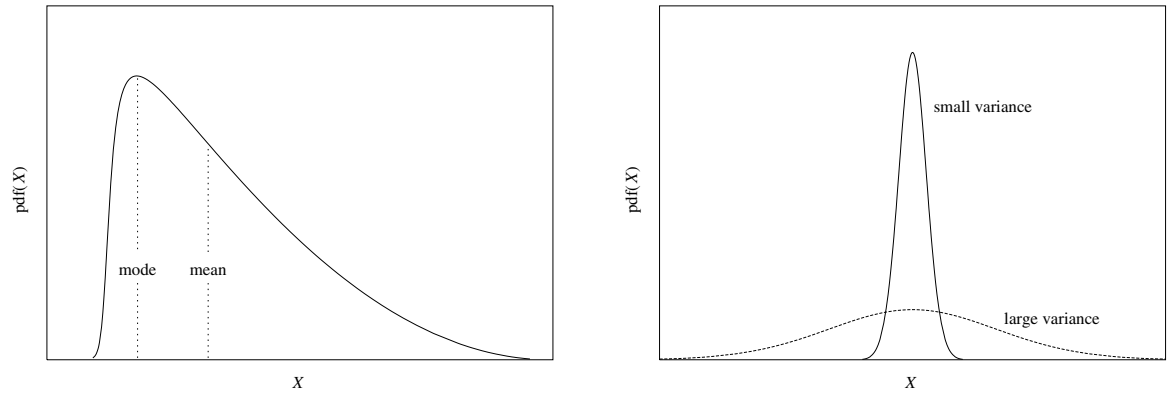

Figure 3.2: Definition of the mean and mode of distribution (left) and probability density functions for large and small variances (right)

Using the binomial theorem, we summarize the relation between the central moments μ_r and the raw moments μ'_r below [104]:

$$\mu_r = \sum_{j=0}^{r} \binom{r}{j} (-1)^{r-j} \mu'_j {\mu'_1}^{r-j}, \tag{3.14}$$

$$\mu'_r = \sum_{j=0}^{r} \binom{r}{j} \mu_j {\mu'_1}^{r-j}, \tag{3.15}$$

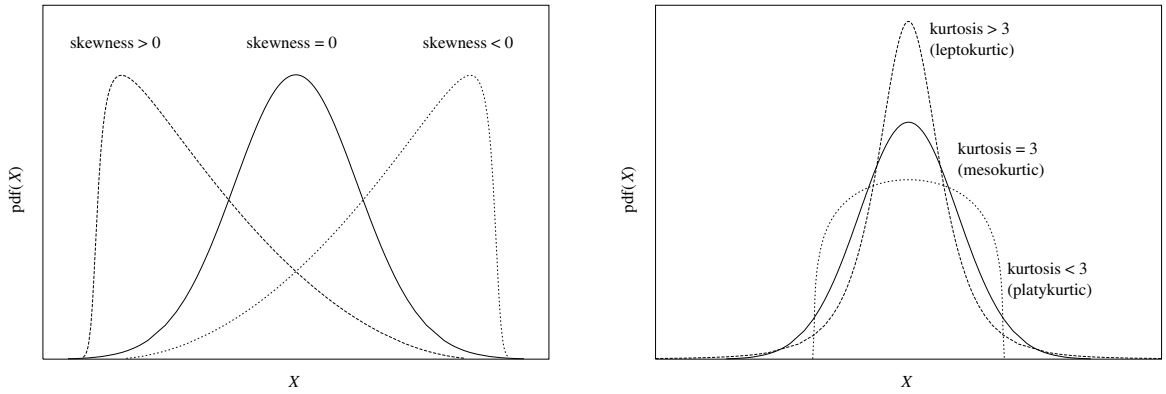

Figure 3.3: Skewness (left), kurtosis (right) and shape of probability density functions

where $\mu_0 = \mu'_0 = 1$ and $\mu_1 = 0$. The relation of the first four moments are therefore

$$\mu_1 = 0, \tag{3.16a}$$
$$\mu_2 = \mu'_2 - {\mu'_1}^2 = \mathsf{Var}[X], \tag{3.16b}$$
$$\mu_3 = \mu'_3 - 3\mu'_1\mu'_2 + 2{\mu'_1}^3, \tag{3.16c}$$
$$\mu_4 = \mu'_4 - 4\mu'_1\mu'_3 + 6{\mu'_1}^2\mu'_2 - 3{\mu'_1}^4, \tag{3.16d}$$

and

$$\mu'_1 = \mu_1, \tag{3.17a}$$
$$\mu'_2 = \mu_2 + \mu_1^2, \tag{3.17b}$$
$$\mu'_3 = \mu_3 + 3\mu_1\mu_2 + \mu_1^3, \tag{3.17c}$$
$$\mu'_4 = \mu_4 + 4\mu_1\mu_3 + 6\mu_1^2\mu_2 + \mu_1^4. \tag{3.17d}$$

Another useful definition for our analysis is conditional expectation. Let X and Y be joint continuous random variables. Then the *conditional expectation* of Y given $X = x$ is defined by

$$\mathsf{E}[Y|x] = \int_{-\infty}^{\infty} y f_Y(y|x)\mathrm{d}y, \tag{3.18}$$

where f_Y is the pdf of Y. Note that $\mathsf{E}[Y|x]$ is the center of mass associated with the conditional pdf. In particular, in our analysis we will be using the following result

$$\begin{aligned}\mathsf{E}[\mathsf{E}[Y|X]] &= \int_{-\infty}^{\infty} \mathsf{E}[Y|x] f_X(x)\mathrm{d}x \\ &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} y f_Y(y|x)\mathrm{d}y f_X(x)\mathrm{d}x \\ &= \int_{-\infty}^{\infty} y \int_{-\infty}^{\infty} f_{X,Y}(x,y)\mathrm{d}x\mathrm{d}y \\ &= \int_{-\infty}^{\infty} y f_Y(y)\mathrm{d}y = \mathsf{E}[Y].\end{aligned} \tag{3.19}$$

The above result also holds for the expected value of a function of Y, i.e.,

$$\mathsf{E}[\mathsf{E}[h(Y)|X]] = \mathsf{E}[h(Y)]. \tag{3.20}$$

3.3.3 Transform Methods

In this section we introduce two transform methods: the moment generating function (mgf) and the probability generating function (pgf). The mgf and pgf will play an important role in our analysis.

The *moment generating function* (mgf) of X is defined by

$$M_X(t) = \mathsf{E}[e^{tX}] \tag{3.21}$$
$$= \begin{cases} \sum_i p_X(x) e^{tx_i} & \text{if } X \text{ is discrete,} \\ \int_{-\infty}^{\infty} f_X(x) e^{tx} \mathrm{d}x & \text{if } X \text{ is continuous.} \end{cases}$$

Although the mgf does not always exist for the problems that we encounter, there will always be an interval of t values in which $M_X(t)$ does exist.

If random variables are a non-negative integer-valued discrete random variables, then it is more convenient to evaluate related problems through the *probability generating function* (pgf), defined by

$$G_X(z) = \mathsf{E}[z^X] \tag{3.22}$$
$$= \sum_{k=0}^{\infty} p_X(k) z^k.$$

The first expression is the expected value of the function of z^X. The second expression is the z-transform of the pmf.

The reason why transform methods are useful lies in the following properties of the transforms. We will give the properties of the mgf, and by appropriate substitution for t, similar properties can be stated for the pgf as well.

Definition 3.1 The convolution theorem [106]
Let $X_1, X_2, \ldots, X_n$ be mutually independent random variables on a given probability space, and let $S_n = \sum_{i=1}^{n} X_i$. If $M_{X_i}(t)$ exists for all i, then $M_{S_n}(t)$ exists, and it holds that

$$M_{S_n}(t) = M_{X_1}(t) M_{X_2}(t) \cdots M_{X_n}(t). \tag{3.23}$$

□

Thus the mgf of a sum of independent random variables is the product of the individual mgf of those variables. The convolution theorem states that we may find the transform of a sum of independent random variables without n-dimensional integration.

From Eq. (3.23), it is easy to show that the mean and variance of S_n are given by

$$\mathsf{E}[S_n] = \mathsf{E}[X_1] + \mathsf{E}[X_2] + \cdots + \mathsf{E}[X_n] \tag{3.24}$$

and

$$\mathsf{Var}[S_n] = \mathsf{Var}[X_1] + \mathsf{Var}[X_2] + \cdots \mathsf{Var}[X_n]. \tag{3.25}$$

The mean of S_n in Eq. (3.24) is equal to the sum of individual means. Since independency between X_is is not a necessary, Eq. (3.24) is generally valid. This property is called the

linearity property of expectation. However, the independent condition is necessary for $\mathsf{Var}[S_n]$, which in general is given by

$$\mathsf{Var}[S_n] = \sum_{k=1}^{n} \mathsf{Var}(X_k) + \sum_{\substack{j=1 \\ j \neq k}}^{n} \sum_{k=1}^{n} \mathsf{Cov}(X_j, X_k), \tag{3.26}$$

where $\mathsf{Cov}(X, Y)$ denotes the covariance between random variables X and Y and is defined by

$$\mathsf{Cov}(X, Y) = \mathsf{E}[(X - \mathsf{E}[X])(Y - \mathsf{E}[Y])]. \tag{3.27}$$

Thus in general, the variance of a sum of random variables is not equal to the sum of the individual variances. The covariance will play an important role in our program analysis and measurements since it reflects the dependency between two random variables.

Since $f_X(x)$ and $M_X(t)$ form a transform pair, it is possible to obtain the moments of X from $M_X(t)$.

Definition 3.2 The moment theorem [58]
The rth raw moments of X are given by

$$\mathsf{E}[X^r] = \frac{\mathrm{d}^r}{\mathrm{d}t^r} M_X(t)\bigg|_{t=0}. \tag{3.28}$$

□

The corresponding property for pgf makes that the pmf of X is given by

$$p_X(r) = \frac{1}{r!} \frac{\mathrm{d}^r}{\mathrm{d}z^r} G_X(z)\bigg|_{z=1}. \tag{3.29}$$

3.4 Analysis

In this section, we present the analysis of sequential, conditional, and parallel task compositions. We explicitly derive the relationship between statistical moments of a program composition in terms of the moments of the constituting components.

As mentioned in Chapter 1, our moment approach is compositional, as illustrated in Figure 3.4. Recall the example in Section 1.1. If the input moments $\mathsf{E}[X^r], \mathsf{E}[N^r]$, and $\mathsf{E}[P^r]$ represent the execution time of basic block, loop bound, and branch probability, respectively, the equivalent output moments $\mathsf{E}[Y^r]$ can be evaluated. In turn, the output moments $\mathsf{E}[Y^r]$ can be used as the input for the embedding control flows related to Y. Hence, the analysis typically proceeds in a bottom-up fashion. Recursive application of the analysis stops when the outermost control flow construct is reached, yielding the moments of the execution time T of the whole program.

As mentioned in Section 3.1 we consider the first four moments as workload representation. The execution time distribution of a basic block, loop bound, branch probability, etc. is given by

$$X = (\mathsf{E}[X], \mathsf{Var}[X], \mathsf{Skw}[X], \mathsf{Kur}[X]), \tag{3.30}$$

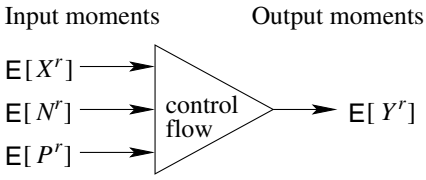

Figure 3.4: Compositional approach.

i.e., the mean, variance, skewness, and kurtosis, respectively. The values can simply be obtained through measurements on a real program or from previous analysis results of the inner control flow compositions.

In our analysis (e.g., see Section 3.4.3), we may encounter a deterministic workload. We define a deterministic program parameter X as

$$X = (\mathsf{E}[X], \mathsf{Var}[X] = 0, \mathsf{Skw}[X] = 0, \mathsf{Kur}[X] = 3). \tag{3.31}$$

3.4.1 Binary Sequential Composition

In this section we present the analysis of binary sequential composition. Consider the following binary sequential composition:

```
statement_1;
statement_2;
```

which specifies a strict sequence of two statements (basic blocks). Let the workload distribution of `statement_i` be denoted by X_i where $i \in \{1, 2\}$. The following proposition describes the execution time distribution of the above sequential composition.

Proposition 3.1 Binary sequential composition
Let Y be defined as an addition of two independent random variables X_1 and X_2,

$$Y = X_1 + X_2, \tag{3.32}$$

where $\mathsf{E}[X_1^r]$ and $\mathsf{E}[X_2^r]$ exist. Then the rth raw moment of Y is given by

$$\mathsf{E}[Y^r] = \sum_{j=0}^{r} \binom{r}{j} \mathsf{E}[X_1^j]\mathsf{E}[X_2^{r-j}]. \tag{3.33}$$

Proof:
The moments of Y can be derived using the mgf of Y as follows:

$$\mathsf{E}[e^{tY}] = \mathsf{E}[e^{t(X_1+X_2)}] = \mathsf{E}[e^{tX_1}]\mathsf{E}[e^{tX_2}]. \tag{3.34}$$

Through expansion of the right-hand side into a power series with respect to t, Eq. (3.34) becomes

$$\mathsf{E}[e^{tY}] = (1 + t\mathsf{E}[X_1] + \frac{t^2\mathsf{E}[X_1^2]}{2!} + \ldots)(1 + t\mathsf{E}[X_2] + \frac{t^2\mathsf{E}[X_2^2]}{2!} + \ldots)$$

$$= 1 + t(\mathsf{E}[X_1] + \mathsf{E}[X_2]) + \ldots + \frac{t^i}{i!}\sum_{j=0}^{i}\binom{i}{j}\mathsf{E}[X_1^j]\mathsf{E}[X_2^{i-j}] + \ldots$$

$$= \sum_{i=0}^{\infty}\frac{t^i}{i!}\sum_{j=0}^{i}\binom{i}{j}\mathsf{E}[X_1^j]\mathsf{E}[X_2^{i-j}]. \tag{3.35}$$

By Eq. (3.28), the rth moment of Y is given by

$$\mathsf{E}[Y^r] = \left.\frac{\mathrm{d}^r\mathsf{E}[e^{tY}]}{\mathrm{d}t^r}\right|_{t=0}.$$

At $t = 0$, the derivative of the series is equal to zero except for $i = r$. Hence, Eq. (3.35) reduces to

$$\mathsf{E}[Y^r] = \sum_{j=0}^{r}\binom{r}{j}\mathsf{E}[X_1^j]\mathsf{E}[X_2^{r-j}].$$

□

A specific moment expression is derived by substituting r. From Eq. (3.33) we have derived the first four moments of Y according to

$$\mathsf{E}[Y] = \mathsf{E}[X_1] + \mathsf{E}[X_2], \tag{3.36a}$$

$$\mathsf{Var}[Y] = \mathsf{Var}[X_1] + \mathsf{Var}[X_2], \tag{3.36b}$$

$$\mathsf{Skw}[Y] = \frac{\mathsf{Skw}[X_1]\mathsf{Std}[X_1]^3 + \mathsf{Skw}[X_2]\mathsf{Std}[X_2]^3}{\mathsf{Std}[Y]^3}, \tag{3.36c}$$

$$\mathsf{Kur}[Y] = \frac{(\mathsf{Kur}[X_1] - 3)\mathsf{Std}[X_1]^4 + (\mathsf{Kur}[X_2] - 3)\mathsf{Std}[X_2]^4}{\mathsf{Std}[Y]^4} + 3. \tag{3.36d}$$

The derivation can be found in Appendix C.1.1.

3.4.2 N-ary Sequential Composition

In the following, we derive the moments for an N-ary sequential composition (a program loop), according to

```
for (i = 1; i <= N; i++)
    statement_i;
```

where X_i, $i \in \{1, 2, \ldots, N\}$, are independent identically distributed (iid) random variables. Note that in addition to X_i, N may also be stochastic, reflecting the fact that many loop bounds are data dependent as well.

Proposition 3.2 N-ary sequential composition
Let Y be defined as the sum of a random number N of random variables X_i,

$$Y = \sum_{i=1}^{N} X_i, \tag{3.37}$$

where X_i are iid variates of random variable X, and N is independent from X_i. Let also $\mathsf{E}[X^r]$ and $\mathsf{E}[N^r]$ exist. Then the rth raw moment of Y is given by

$$\mathsf{E}[Y^r] = \mathsf{E}\left[\frac{\mathrm{d}^r}{\mathrm{d}t^r}\left(\sum_{j=0}^{r}\frac{t^j\mathsf{E}[X^j]}{j!}\right)^N\bigg|_{t=0}\right]. \tag{3.38}$$

Proof:
The moment generating function of Y can be found using conditional expectation in Eqs. (3.19) and (3.20):

$$\mathsf{E}[e^{tY}] = \mathsf{E}[\mathsf{E}[e^{tY}|N]]. \tag{3.39}$$

For $N = n$, we have

$$\begin{aligned}\mathsf{E}[e^{tY}|n] &= \mathsf{E}[e^{t(X_1+\cdots+X_n)}]\\ &= \mathsf{E}[e^{tX_1}]\ldots\mathsf{E}[e^{tX_n}]\\ &= \prod_{i=1}^{n}\mathsf{E}[e^{tX_i}].\end{aligned} \tag{3.40}$$

As $X_i = X$, Eq. (3.39) becomes

$$\mathsf{E}[e^{tY}|N] = \mathsf{E}[e^{tX}]^N. \tag{3.41}$$

By Eq. (3.20), it follows that

$$\mathsf{E}[e^{tY}] = \mathsf{E}[\mathsf{E}[e^{tX}]^N]. \tag{3.42}$$

The rth moment of Y is given by

$$\mathsf{E}[Y^r] = \frac{\mathrm{d}^r\mathsf{E}[e^{tY}]}{\mathrm{d}t^r}\bigg|_{t=0} = \frac{\mathrm{d}^r\mathsf{E}[\mathsf{E}[e^{tX}]^N]}{\mathrm{d}t^r}\bigg|_{t=0}.$$

Using the linearity property of expectation, and expanding $\mathsf{E}[e^{tX}]$ into a power series with respect to t (similar to the proof in Proposition 3.1), we obtain

$$\mathsf{E}[Y^r] = \mathsf{E}\left[\frac{\mathrm{d}^r}{\mathrm{d}t^r}\left(\sum_{j=0}^{\infty}\frac{t^j\mathsf{E}[X^j]}{j!}\right)^N\bigg|_{t=0}\right]. \tag{3.43}$$

Since the derivative of the jth term of the series is equal to zero at $t = 0$ for $j > r$, Eq. (3.43) reduces to

$$\mathsf{E}[Y^r] = \mathsf{E}\left[\frac{\mathrm{d}^r}{\mathrm{d}t^r}\left(\sum_{j=0}^{r}\frac{t^j\mathsf{E}[X^j]}{j!}\right)^N\bigg|_{t=0}\right]. \tag{3.44}$$

□

A specific moment expression is derived by substituting r. For instance, for the first four moments we have derived the following results

$$\mathsf{E}[Y] = \mathsf{E}[N]\mathsf{E}[X], \tag{3.45a}$$

$$\mathsf{Var}[Y] = \mathsf{E}[N]\mathsf{Var}[X] + \mathsf{E}[X]^2\mathsf{Var}[N], \tag{3.45b}$$

$$\begin{aligned}\mathsf{Skw}[Y] = {} & (\mathsf{E}[N]\mathsf{Skw}[X]\mathsf{Std}[X]^3 + \mathsf{E}[X]^3\mathsf{Skw}[N]\mathsf{Std}[N]^3 \\ & + 3\mathsf{E}[X]\mathsf{Var}[X]\mathsf{Var}[N])/\mathsf{Std}[Y]^3,\end{aligned} \tag{3.45c}$$

$$\begin{aligned}\mathsf{Kur}[Y] = {} & (\mathsf{E}[N]\mathsf{Kur}[X]\mathsf{Var}[X]^2 + \ \mathsf{E}[X]^4\mathsf{Kur}[N]\mathsf{Var}[N]^2 \\ & + 6\mathsf{E}[X]^2\mathsf{Var}[X]\mathsf{Skw}[N]\mathsf{Std}[N]^3 + 4\mathsf{E}[X]\mathsf{Var}[N]\mathsf{Skw}[X]\mathsf{Std}[X]^3 \\ & + 3\mathsf{Var}[X]^2\mathsf{Var}[N])/\mathsf{Var}[Y]^2.\end{aligned} \tag{3.45d}$$

The derivation can be found in Appendix C.1.2.

3.4.3 Binary Conditional Composition

In this section we describe the analysis of the binary conditional composition

```
if (C)
    statement_1;
else
    statement_2;
```

with branch condition `C`. If `C` is true, `statement_1` will be executed, otherwise `statement_2`. Note that we include the optional `else` clause for completeness of our analysis (a simple `if`-statement is modeled by assigning the workload of `statement_2` to zero). The execution time distribution of the branch is expressed by the following proposition.

Proposition 3.3 Binary conditional composition
Let Y be defined as

$$Y = \begin{cases} X_1, & \text{if } \mathtt{C} = \text{true} \\ X_2, & \text{if } \mathtt{C} = \text{false}, \end{cases} \tag{3.46}$$

where `C` *and X_i are independent. Let* `C` *have truth and false probability P and $Q = 1 - P$, respectively, and let $\mathsf{E}[P^r]$, $\mathsf{E}[Q^r]$ and $\mathsf{E}[X_i^r]$ exist. Then the rth raw moment of Y is given by*

$$\mathsf{E}[Y^r] = \mathsf{E}\left[\frac{\mathrm{d}^r}{\mathrm{d}t^r}\left(\sum_{j=0}^{r}\frac{t^j\mathsf{E}[X_1^j]}{j!}\right)^P\Bigg|_{t=0}\right] + \mathsf{E}\left[\frac{\mathrm{d}^r}{\mathrm{d}t^r}\left(\sum_{j=0}^{r}\frac{t^j\mathsf{E}[X_2^j]}{j!}\right)^Q\Bigg|_{t=0}\right]. \tag{3.47}$$

Proof:
The moments of Y can be derived using conditional expectation based on the mgf of Y:

$$\begin{aligned}\mathsf{E}[e^{tY}] &= \mathsf{E}[\mathsf{E}[e^{tX_1}|\mathtt{C} = \text{true}]] + \mathsf{E}[\mathsf{E}[e^{tX_2}|\mathtt{C} = \text{false}]] \\ &= \mathsf{E}[\mathsf{E}[e^{tX_1}|P]] + \mathsf{E}[\mathsf{E}[e^{tX_2}|Q]].\end{aligned} \tag{3.48}$$

By Eq. (3.42), it follows that

$$\mathsf{E}[e^{tY}] = \mathsf{E}[\mathsf{E}[e^{tX_1}]^P] + \mathsf{E}[\mathsf{E}[e^{tX_2}]^Q]. \tag{3.49}$$

The rth moment of Y is given by

$$\begin{aligned}\mathsf{E}[Y^r] &= \left.\frac{\mathrm{d}^r \mathsf{E}[e^{tY}]}{\mathrm{d}t^r}\right|_{t=0} \\ &= \left.\frac{\mathrm{d}^r \mathsf{E}[\mathsf{E}[e^{tX_1}]^P]}{\mathrm{d}t^r}\right|_{t=0} + \left.\frac{\mathrm{d}^r \mathsf{E}[\mathsf{E}[e^{tX_2}]^Q]}{\mathrm{d}t^r}\right|_{t=0}.\end{aligned} \tag{3.50}$$

Using the linearity property of expectation, and expanding $\mathsf{E}[e^{tX}]$ into a power series with respect to t, we obtain

$$\mathsf{E}[Y^r] = \mathsf{E}\left[\left.\frac{\mathrm{d}^r}{\mathrm{d}t^r}\left(\sum_{j=0}^{\infty}\frac{t^j\mathsf{E}[X_1^j]}{j!}\right)^P\right|_{t=0}\right] + \mathsf{E}\left[\left.\frac{\mathrm{d}^r}{\mathrm{d}t^r}\left(\sum_{j=0}^{\infty}\frac{t^j\mathsf{E}[X_2^j]}{j!}\right)^Q\right|_{t=0}\right]. \tag{3.51}$$

Since the derivative of the jth term of the series is equal to zero at $t = 0$ for $j > r$, Eq. (3.51) reduces to

$$\mathsf{E}[Y^r] = \mathsf{E}\left[\left.\frac{\mathrm{d}^r}{\mathrm{d}t^r}\left(\sum_{j=0}^{r}\frac{t^j\mathsf{E}[X_1^j]}{j!}\right)^P\right|_{t=0}\right] + \mathsf{E}\left[\left.\frac{\mathrm{d}^r}{\mathrm{d}t^r}\left(\sum_{j=0}^{r}\frac{t^j\mathsf{E}[X_2^j]}{j!}\right)^Q\right|_{t=0}\right].$$

□

Note that the random variables P and Q are correlated linearly, i.e., $Q = 1 - P$. Consequently,

$$\mathsf{E}[Q] = 1 - \mathsf{E}[P],\ \mathsf{Var}[Q] = \mathsf{Var}[P],\ \mathsf{Skw}[Q] = -\mathsf{Skw}[P] \text{ and } \mathsf{Kur}[Q] = \mathsf{Kur}[P]. \tag{3.52}$$

From Eq. (3.47) we have derived the first four moments of Y according to

$$\mathsf{E}[Y] = \mathsf{E}[P]\mathsf{E}[X_1] + \mathsf{E}[Q]\mathsf{E}[X_2], \tag{3.53a}$$

$$\begin{aligned}\mathsf{Var}[Y] = {} & \mathsf{E}[P]\mathsf{Var}[X_1] + \mathsf{Var}[P]\mathsf{E}[X_1]^2 + \mathsf{E}[Q]\mathsf{Var}[X_2] + \mathsf{Var}[Q]\mathsf{E}[X_2]^2 \\ & - 2\mathsf{E}[P]\mathsf{E}[Q]\mathsf{E}[X_1]\mathsf{E}[X_2],\end{aligned} \tag{3.53b}$$

$$\begin{aligned}\mathsf{Skw}[Y] = {} & (\mathsf{E}[P]\mathsf{Skw}[X_1]\mathsf{Std}[X_1]^3 + \mathsf{E}[X_1]^3\mathsf{Skw}[P]\mathsf{Std}[P]^3 + 3\mathsf{Var}[P]\mathsf{E}[X_1]\mathsf{Var}[X_1] \\ & + \mathsf{E}[Q]\mathsf{Skw}[X_2]\mathsf{Std}[X_2]^3 + \mathsf{E}[X_2]^3\mathsf{Skw}[Q]\mathsf{Std}[Q]^3 + 3\mathsf{Var}[Q]\mathsf{E}[X_2]\mathsf{Var}[X_2] \\ & - 3\mathsf{E}[P]\mathsf{E}[X_1](\mathsf{E}[Q]\mathsf{Var}[X_2] + (\mathsf{Var}[Q] - \mathsf{E}[Q]^2)\mathsf{E}[X_2]^2) \\ & - 3\mathsf{E}[Q]\mathsf{E}[X_2](\mathsf{E}[P]\mathsf{Var}[X_1] + (\mathsf{Var}[P] - \mathsf{E}[P]^2)\mathsf{E}[X_1]^2))/\mathsf{Std}[Y]^3,\end{aligned} \tag{3.53c}$$

$$\begin{aligned}\mathsf{Kur}[Y] = {} & \mathsf{E}[P]\mathsf{Kur}[X_1]\mathsf{Var}[X_1]^2 + \mathsf{E}[Q]\mathsf{Kur}[X_2]\mathsf{Var}[X_2]^2 \\ & + 4\mathsf{E}[P^2]\mathsf{E}[X_1]\mathsf{Skw}[X_1]\mathsf{Std}[X_1]^3 + 4\mathsf{E}[Q^2]\mathsf{E}[X_2]\mathsf{Skw}[X_2]\mathsf{Std}[X_2]^3 \\ & + \mathsf{E}[X_1]^4((\mathsf{Kur}[P] + 3)\mathsf{Var}[P]^2 + 4\mathsf{E}[P]\mathsf{Skw}[P]\mathsf{Std}[P]^3 + 6\mathsf{E}[P]^2\mathsf{Var}[P] + \mathsf{E}[P]^4) \\ & + \mathsf{E}[X_2]^4((\mathsf{Kur}[Q] + 3)\mathsf{Var}[Q]^2 + 4\mathsf{E}[Q]\mathsf{Skw}[Q]\mathsf{Std}[Q]^3 + 6\mathsf{E}[Q]^2\mathsf{Var}[Q] + \mathsf{E}[Q]^4) \\ & + 6\mathsf{E}[X_1]^2\mathsf{Var}[X_1](\mathsf{Skw}[P]\mathsf{Std}[P]^3 + 3\mathsf{E}[P]\mathsf{Var}[P] + \mathsf{E}[P]^3) \\ & + 6\mathsf{E}[X_2]^2\mathsf{Var}[X_2](\mathsf{Skw}[Q]\mathsf{Std}[Q]^3 + 3\mathsf{E}[Q]\mathsf{Var}[Q] + \mathsf{E}[Q]^3) \\ & + 3(\mathsf{E}[P] + \mathsf{E}[P]^2 + \mathsf{Var}[P])\mathsf{Var}[X_1]^2 + 3(\mathsf{E}[Q] + \mathsf{E}[Q]^2 + \mathsf{Var}[Q])\mathsf{Var}[X_2]^2\end{aligned} \tag{3.53d}$$

Eq. (3.53) expresses the output moments Y of the conditional composition in terms of the input moments X_i, P and Q. The derivation can be found in Appendix C.2.1.

Like in sequential composition, it is straightforward to model the workload distribution X_i. However, modeling the truth probability P (also Q) in conditional composition is far from trivial. The issues concerning statistical branch modeling will be discussed in depth in Chapter 4.

3.4.4 N-ary Conditional Composition

Consider the following N-ary conditional composition

```
if (C_1)
    statement_1;
elseif (C_2)
    statement_2;
...
elseif (C_N)
    statement_N;
```

In the composition, only one condition C_i is assumed to be true. The analysis is given by the following proposition.

Proposition 3.4 N-ary conditional composition
Let Y be defined as

$$Y = \begin{cases} X_1, & \text{if } C_1 = \text{true}, \\ X_2, & \text{if } C_2 = \text{true}, \\ \vdots & \vdots \\ X_N, & \text{if } C_N = \text{true} \end{cases} \tag{3.54}$$

where C_i and X_i are independent, while C_i are mutually exclusive. Let C_i have truth probabilities P_i, and let $\mathsf{E}[P_i^r]$ and $\mathsf{E}[X_i^r]$ exist. Then the rth raw moment of Y is given by

$$\mathsf{E}[Y^r] = \sum_{i=1}^{N} \mathsf{E}\left[\frac{\mathrm{d}^r}{\mathrm{d}t^r} \left(\sum_{j=0}^{r} \frac{t^j \mathsf{E}[X_i^j]}{j!} \right)^{P_i} \Bigg|_{t=0} \right]. \tag{3.55}$$

Proof:
We prove Proposition 3.4 by induction.
(1) For $i = 2$, we have proven Eq. (3.55) in Proposition 3.3.
(2) Assume that (1) holds for $i = n$, that is,

$$\mathsf{E}[Y_n^r] = \sum_{i=1}^{n} \mathsf{E}\left[\frac{\mathrm{d}^r}{\mathrm{d}t^r} \left(\sum_{j=0}^{r} \frac{t^j \mathsf{E}[X_i^j]}{j!} \right)^{P_i} \Bigg|_{t=0} \right]. \tag{3.56}$$

We must now show that it also holds for $i = n + 1$. Since in Eq. (3.56) it holds that $\sum_{i=1}^{n} P_i = 1$, we let $P_{n+1} = 0$. It follows that

$$\mathsf{E}[Y_{n+1}^r] = \sum_{i=1}^{n} \mathsf{E}\left[\frac{\mathrm{d}^r}{\mathrm{d}t^r} \left(\sum_{j=0}^{r} \frac{t^j \mathsf{E}[X_i^j]}{j!} \right)^{P_i} \Bigg|_{t=0} \right] + \mathsf{E}\left[\frac{\mathrm{d}^r}{\mathrm{d}t^r} \left(\sum_{j=0}^{r} \frac{t^j \mathsf{E}[X_{n+1}^j]}{j!} \right)^{P_{n+1}} \Bigg|_{t=0} \right]$$

$$= \sum_{i=1}^{n+1} \mathsf{E}\left[\frac{\mathrm{d}^r}{\mathrm{d}t^r} \left(\sum_{j=0}^{r} \frac{t^j \mathsf{E}[X_i^j]}{j!} \right)^{P_i} \Bigg|_{t=0} \right] \tag{3.57}$$

due to the linearity property of expectation. Eq. (3.57) still holds for $P_{n+1} \neq 0$, provided that $\sum_{i=1}^{n+1} P_i = 1$.

□

Unlike X_i, the statistical moments of P are very difficult to obtain in practice. As mentioned earlier, the problems related to statistical branch modeling will be discussed in depth in Chapter 4.

3.4.5 Parallel Composition

Parallel composition can be divided in *and*-parallel and *or*-parallel compositions. In and-parallelism, the parallel composition has finished when *all* constituent tasks have finished (i.e., the last task). In or-parallelism, the parallel composition has finished when *any* constituent task has finished (i.e., the first task). Of the two forms, and-parallel composition is most common in parallel computing, and typically results from *task* or *data* parallelism, where each task essentially involves *different* computation (workload), or the tasks involve the *same* computation (on different data), respectively.

Or-parallel composition, in contrast, results from *speculative* parallelism, where each task is initiated without being sure that its execution needs to complete. Examples of or-parallel composition include searching in parallel for a solution, such as finding a unique pattern in a data base, where the event of the solution found terminates the entire parallel process.

Consider the following and-parallel composition:

```
forall (i = 1; i <= N; i++)
    statement_i;
```

which comprises N processes running in parallel without any intermediate form of synchronization. For simplicity, let `statement_i` have an iid workload distribution, denoted by X_i. Since in and-parallelism the slowest computation (largest workload) determines the overall execution time, the execution time Y is defined as

$$Y = \max(X_1, X_2, \ldots, X_N). \tag{3.58}$$

Unlike in the sequential domain, in parallel composition, there is in general no analytic, closed-form solution for $\mathsf{E}[Y^r]$ in terms of $\mathsf{E}[X^r]$ due to the following *integration problem*. From Eqs. (2.7) and (3.9), we obtain

$$\begin{aligned} \mathsf{E}[Y^r] &= \int_{-\infty}^{\infty} x^r \mathrm{d}F_Y(x) \\ &= \int_{-\infty}^{\infty} x^r \mathrm{d}(F_X(x))^N \\ &= N \int_{-\infty}^{\infty} x^r (F_X(x))^{N-1} \mathrm{d}F_X(x) \end{aligned} \tag{3.59}$$

The right-hand side integration is in general fundamentally impossible to solve analytically in such a way that $\mathsf{E}[Y^r]$ can be expressed directly as function of $\mathsf{E}[X^r]$. The same problem as in Eq. (3.59) also applies for or-parallel composition. In this thesis, we present an approximate solution to this problem, which is described in Chapter 5.

3.5 Examples

While our approach to conditional and parallel compositions will be elaborated in Chapters 4 and 5, respectively, in this section, we apply our approach to three simple sequential codes in order to illustrate how our moment analysis is used to predict real program performance. Three small example codes are analyzed: Vector Scaling, Straight Selection Sort, and a version of Quicksort. In Straight Selection Sort we focus on control flow structures while in Quicksort we focus on the stochastic basic block execution times due to the effect of memory hierarchy. The choice of applications was based on two considerations. The first is a high data set dependency of the programs, as we are interested in analyzing stochastic program behavior. The second is the simplicity of the programs so that we could fully instrument and analyze the programs by hand. In order to avoid measurements on real machines while validating our analysis technique, we typically modeled a basic block by a deterministic unit time, which allows a simple counter-based measurement technique for T. This would imply that all X_i are deterministic. However, since the programs typically involve multiple compositions, a higher-level composition typically involves a stochastic workload. For example, consider a loop statement with a stochastic loop bound N. Although X_i are deterministic, from Eq. (3.45) we can verify that Y is stochastic due to N. Thus, the approach can be applied without loss of generality. Of course, if the actual moments of the real basic-block execution times, X_i, were known in advance, these values could be incorporated in the analysis in a trivial manner. To compare the prediction with actual program performance, we carried out measurements where every basic block in the program simply increments a unit counter whose final value represents the measured execution time T.

Typically, when analyzing loops, one assumes statistical independence between iterations, while often characterizing branches in terms of a deterministic truth probability, thus ignoring the fact that a condition will not always behave like a Bernoulli process. Apart from validating our analysis method, we will also study to what extent such assumptions actually hold.

3.5.1 Vector Scaling

An implementation of Vector Scaling is given by

```
for (i = 1; i <= n; i++)
    if (x[i] != 0)
        x[i] = x[i] * alpha;
```

which scales a sparse double-precision vector x of length n. Let the execution time be given by the double-precision multiplication time τ, ignoring other program contributions. Let p denote the truth probability of the branch condition `(x[i] != 0)`.

Let n and τ be deterministic, and let the branch condition be modeled by a Bernoulli distribution with parameter p. Let the random variable P denote the branch probability. For a Bernoulli process, the first four moments of P are given by [106]

$$\mathsf{E}[P] = p, \tag{3.60a}$$
$$\mathsf{Var}[P] = p(1-p), \tag{3.60b}$$
$$\mathsf{Skw}[P] = p(1-p)(1-2p)/\mathsf{Std}[P]^3, \tag{3.60c}$$
$$\mathsf{Kur}[P] = p(1-p)(1-3p+3p^2)/\mathsf{Std}[P]^4. \tag{3.60d}$$

As the moment approach can be applied in a bottom-up fashion, we begin by analyzing the conditional composition. Let the execution time be denoted by X. From Eqs. (3.53) and (3.60), we obtain (τ deterministic)

$$\mathsf{E}[X] = p\tau, \tag{3.61a}$$
$$\mathsf{Var}[X] = p(1-p)\tau^2, \tag{3.61b}$$
$$\mathsf{Skw}[X] = p(1-p)(1-2p)\tau^3/\mathsf{Std}[X]^3, \tag{3.61c}$$
$$\mathsf{Kur}[X] = p(1-p)(1-3p+3p^2)\tau^4/\mathsf{Std}[X]^4. \tag{3.61d}$$

Note that since the else-clause of the conditional composition is empty, we can assign X_2 in Eq. (3.53) to zero. In the last step, we apply N-ary sequential composition using Eq. (3.45). By substituting Eq. (3.61) in Eq. (3.45), our analysis yields the following prediction (n deterministic) :

$$\mathsf{E}[T] = np\tau, \tag{3.62a}$$
$$\mathsf{Var}[T] = np(1-p)\tau^2, \tag{3.62b}$$
$$\mathsf{Skw}[T] = np(1-p)(1-2p)\tau^3/\mathsf{Std}[T]^3, \tag{3.62c}$$
$$\mathsf{Kur}[T] = (np(1-p) - 6np^2(1-p)^2 + 3(np)^2(1-p)^2)\tau^4/\mathsf{Std}[T]^4. \tag{3.62d}$$

The Vector Scaling code was measured for $n = 1,000$ and 5,000 random data sets `x`. The elements of vector `x` are generated according to a Bernoulli distribution with probability $p = 0.1$ (p true $\rightarrow x[i] \neq 0$) to resemble data sets with average density $d = 0.1$. The moments of n and P are given in Table 3.1 The measured probability mass function (pmf)

Table 3.1: n and $\mathsf{E}[P^r]$ for Vector Scaling

X	$\mathsf{E}[X]$	$\mathsf{Var}[X]$	$\mathsf{Skw}[X]$	$\mathsf{Kur}[X]$
n	10,000	0	0	3
P	0.1	$9\ 10^{-2}$	2.7	8.1

of T is given by the histogram in Figure 3.5. In the figure, the pmf of T is compared with the predicted execution time distribution corresponding to $\mathsf{E}[T] = 100, \mathsf{Var}[T] = 90, \mathsf{Skw}[T] = 0$ and $\mathsf{Kur}[T] = 3$, as predicted according to Eqs. (3.62b) to (3.62d) with

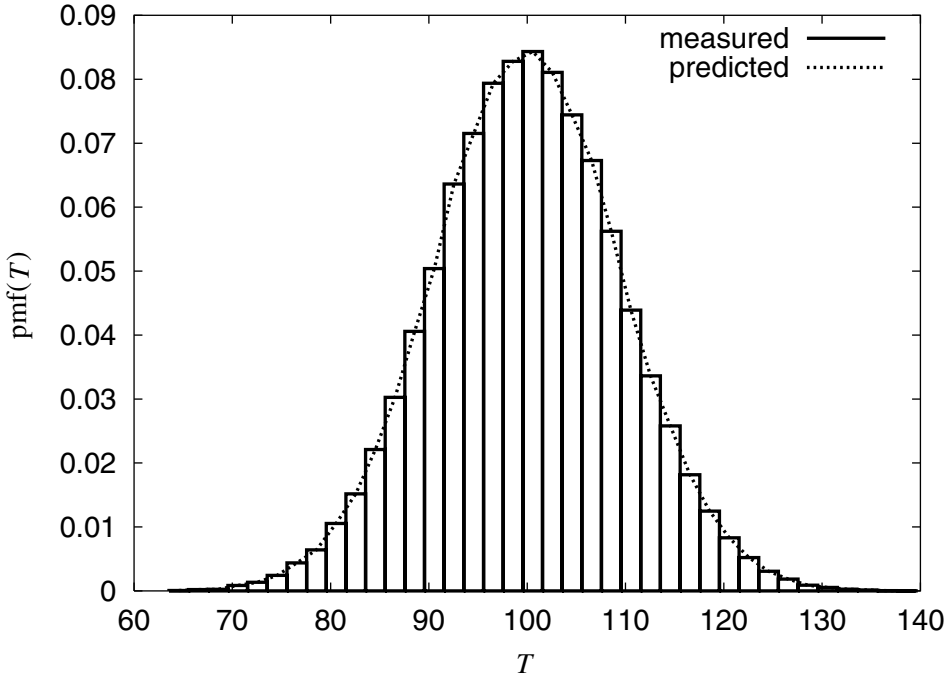

Figure 3.5: The execution time of array scaling.

$n = 1,000, p = d, \tau = 1$. The agreement between measurement and prediction shows that in this example the Bernoulli assumption used in deriving Eqs. (3.62b) to (3.62d) is indeed valid. This result is not surprising since the input array elements are generated with Bernoulli distributed random variables. In the next examples we will show that branch conditions cannot be modeled by a Bernoulli distribution in all cases.

3.5.2 Straight Selection Sort

In this section, we apply our analysis technique to Straight Selection Sort. Our discussion is close to that in [106] where a symbolic formula for the sorting program is derived by *hand*. Although in [106] also a closed-form symbolic formula for the execution time is derived, our method is directly based on *mechanically* deriving the overall execution time distribution based on the *control structure* of the program.

Consider the following Straight Selection Sort implementation

```
for (i=N; i>=2; i--) {
    k = i;
    for (j=i-1; j>=1; j--)
        if (a[j]>a[k])
            k = j;
        swap(a[i],a[k]);
}
```

where `swap` exchanges the value between two elements. The code comprises a two-fold nested loop and a conditional statement in the innermost loop. The execution frequency

of the statements `swap` and `k = i` is deterministic and equals $N-1$, while the execution frequency of `k = j` is stochastic. To define program performance, we choose the statement `k = j` to take a unit time execution time delay while the rest of the constructs are assumed to take zero execution time. Although in practice the statement `k = j` will not represent any significant execution time, this setup provides the most interesting scenario since the sorting program yields stochastic execution time.

We analyze the program in a bottom-up fashion. First, we study the innermost `for` loop, which we call `Max`. Let the branch condition `a[j]>a[k]` be modeled by a Bernoulli process with parameter p. If the branch probability is denoted by P, the first four moments are given by Eq. (3.60). The analysis of the `Max` loop is similar to the analysis of the vector scaling program in the previous example. It follows that the first four moments of $T_{\texttt{Max}}$ are given by

$$\mathsf{E}[T_{\mathrm{Max},i}] = (i-1)\mathsf{E}[P] = (i-1)p, \tag{3.63a}$$

$$\mathsf{Var}[T_{\mathrm{Max},i}] = (i-1)\mathsf{Var}[P] = (i-1)p(1-p), \tag{3.63b}$$

$$\begin{aligned}\mathsf{Skw}[T_{\mathrm{Max},i}] &= (i-1)\mathsf{Skw}[P]\mathsf{Std}[P]^3/\mathsf{Std}[T_{\mathrm{Max},i}]^3 \\ &= (i-1)p(1-p)(1-2p)/\mathsf{Std}[T_{\mathrm{Max},i}]^3,\end{aligned} \tag{3.63c}$$

$$\begin{aligned}\mathsf{Kur}[T_{\mathrm{Max},i}] &= (i-1)(\mathsf{Kur}[P]-3)\mathsf{Std}[P]^4/\mathsf{Std}[T_{\mathrm{Max},i}]^4 \\ &= ((i-1)p(1-p) - 6(i-1)p^2(1-p)^2 \\ &\quad + 3((i-1)p)^2(1-p)^2)/\mathsf{Std}[T_{\mathrm{Max},i}]^4.\end{aligned} \tag{3.63d}$$

To study the validity of the Bernoulli assumption, we show in Figure 3.6 a plot of the predicted $T_{\texttt{Max}}$ for $i = 1000$, where $\mathsf{E}[P] = 6.44 \times 10^{-3}$ as measured from 5,000 random data sets. The input data set is modeled by an array with length i, where each element is independently generated according to a continuous uniform distribution with sample space $[0, 1]$. To compare the predicted execution time with the measured T_{Max}, shown in Figure 3.6 by the histogram, we have used a Chi-square test [21]. Although the distributions seem alike, the results of this test as shown in Table 3.2 indicates that the branch conditional in `Max` cannot be modeled by a Bernoulli process (due to the assignment `k=j`). This is also demonstrated by Figure 3.7, which shows the measured first four moments as a function of i. Although the skewness and kurtosis are approximately equal to 0 and 3, respectively, the mean and variance increase logarithmically as a function of i. This implies that the average probability p is a *function* of i rather than a constant. Furthermore, the measured values are far from our results, which predict that the mean and variance are linear functions of i.

Next, we continue with the complete analysis of Straight Selection Sort. Based on the fact that the average probability p is a function of i rather than a constant, we now account for *all* moments associated with the previous branch condition through *measurement* of $\mathsf{E}[P^r]$ rather than by just predicting $\mathsf{E}[P] = p$ by assuming Bernoulli behavior[1]. Since X_i depends (linearly) on i, we use Eq. (3.36) rather than Eq. (3.45), since a dependency exists between the inner and outer loops. The last step is analyzing the outer loop. From

[1]The issues involved in branch modeling are discussed in depth in Chapter 4.

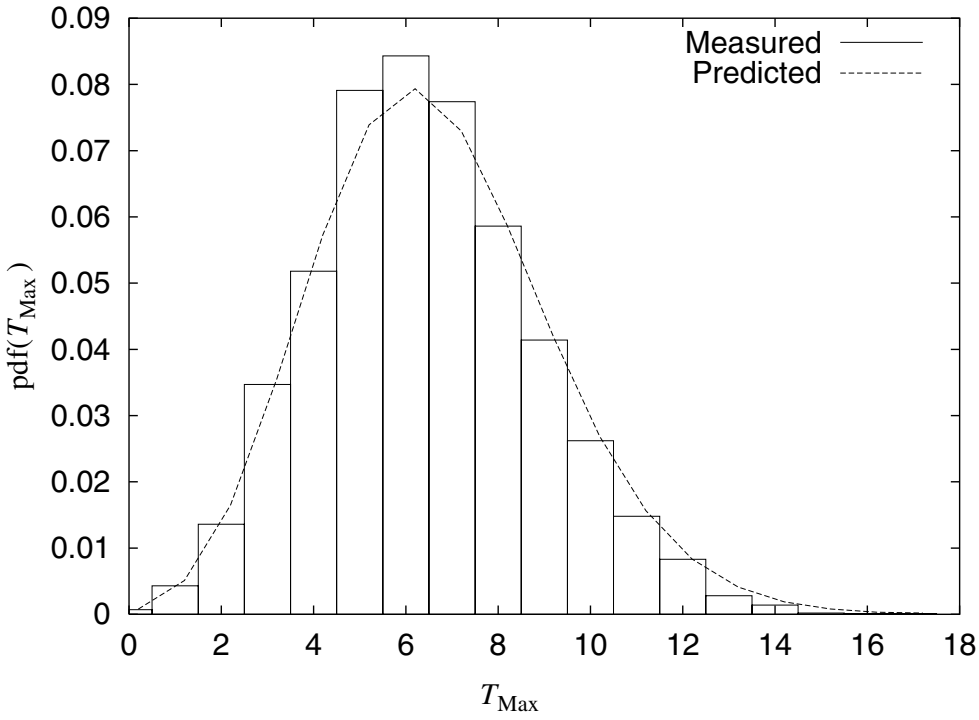

Figure 3.6: The pmf of T_{Max}

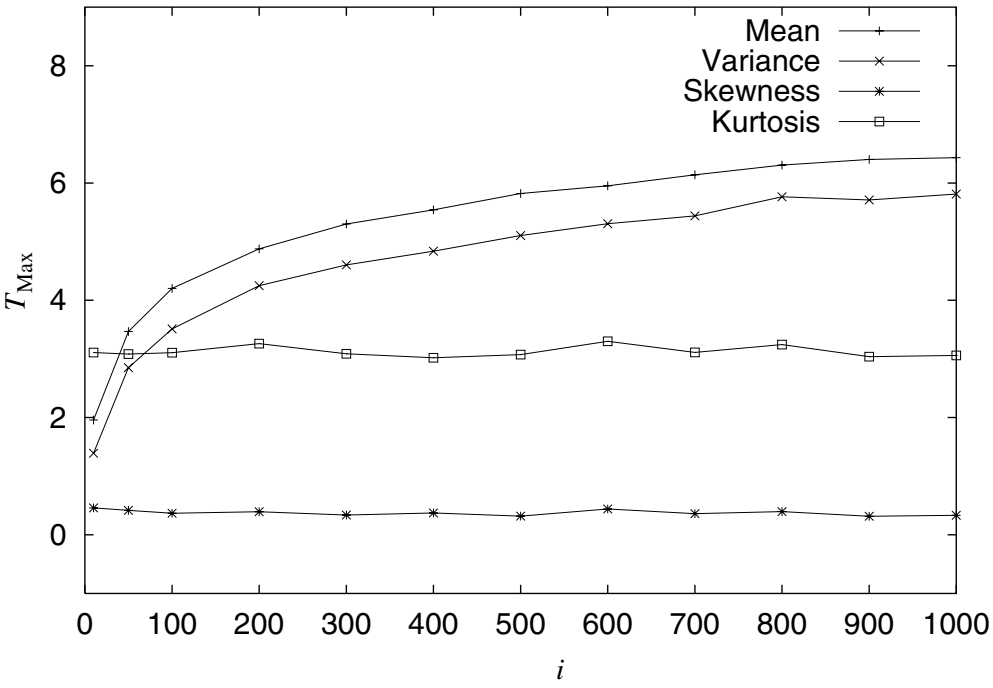

Figure 3.7: The first four moments of T_{Max}

Table 3.2: A Chi-square test for the pmf of `Max` to the normal distribution

Interval	Observed	Expected	$(O-E)^2/E$
0	7	7.87	0.1
1	43	50.95	1.24
2	136	164.79	5.03
3	347	354.97	0.18
4	518	572.90	5.26
5	791	738.97	3.66
6	843	793.51	3.09
7	774	729.62	2.70
8	586	586.42	0.00
9	414	418.54	0.05
10	262	268.57	0.16
11	148	156.51	0.46
12	83	83.53	0.00
13	28	41.10	3.56
14	14	18.76	1.21
15	2	7.99	3.11
16	1	3.18	1.50
≥ 17	1	1.82	0.37
Chi-Square Value		=	31.69
Threshold		=	28.87

Eqs. (3.36) and (3.63) our analysis yields

$$\mathsf{E}[T] = \sum_{i=2}^{N}(i-1)\mathsf{E}[P], \tag{3.64a}$$

$$\mathsf{Var}[T] = \sum_{i=2}^{N}(i-1)\mathsf{Var}[P], \tag{3.64b}$$

$$\mathsf{Skw}[T] = \sum_{i=2}^{N}(i-1)\mathsf{Skw}[P]\mathsf{Std}[P]^3/\mathsf{Std}[T]^3, \tag{3.64c}$$

$$\mathsf{Kur}[T] = \sum_{i=2}^{N}(i-1)(\mathsf{Kur}[P]-3)\mathsf{Std}[P]^4/\mathsf{Std}[T]^4. \tag{3.64d}$$

When mathematical reduction is applied, Eq. (3.64) reduces to

$$\mathsf{E}[T] = \mathsf{E}[P]\sum_{i=2}^{N}(i-1) = N(N-1)\mathsf{E}[P]/2, \tag{3.65a}$$

$$\mathsf{Var}[T] = \mathsf{Var}[P]\sum_{i=2}^{N}(i-1) = N(N-1)\mathsf{Var}[P]/2, \tag{3.65b}$$

$$\begin{aligned}
\mathsf{Skw}[T] &= \mathsf{Skw}[P]\mathsf{Std}[P]^3 \sum_{i=2}^{N}(i-1)/\mathsf{Std}[T]^3 \\
&= \mathsf{Skw}[P]\mathsf{Std}[P]^3(N(N-1)/2)/(N(N-1)\mathsf{Var}[P]/2)^{3/2} \\
&= \sqrt{2}\mathsf{Skw}[P]/\sqrt{N(N-1)}, \qquad (3.65c) \\
\mathsf{Kur}[T] &= (\mathsf{Kur}[P]-3)\mathsf{Std}[P]^4 \sum_{i=2}^{N}(i-1)/\mathsf{Std}[T]^4 \\
&= (\mathsf{Kur}[P]-3)\mathsf{Std}[P]^4(N(N-1)/2)/(N(N-1)\mathsf{Var}[P]/2)^2 \\
&= 2(\mathsf{Kur}[P]-3)/(N(N-1)). \qquad (3.65d)
\end{aligned}$$

Unlike Eq. (3.64), Eq. (3.65) has $O(1)$ time complexity.

For $N = 1,000$ the moments of P obtained from measurements are $\mathsf{E}[P] = 1.1 \times 10^{-2}, \mathsf{Var}[P] = 1.7 \times 10^{-2}, \mathsf{Skw}[P] = 4.4 \times 10^{-3}$, and $\mathsf{Kur}[P] = 3$. Figure 3.8 shows the pmf of the Straight Selection Sort execution time. We compare the measured pmf with the predicted pmf as shown by Figure 3.8. As illustrated in the figure, the mean is predicted quite accurately. However, the predicted execution time has a smaller variance than the observed one, which suggests that a correlation must exist between individual `Max` invocations in each iteration of the Straight Selection Sort i-loop, resulting in non-zero covariance between individual iterations.

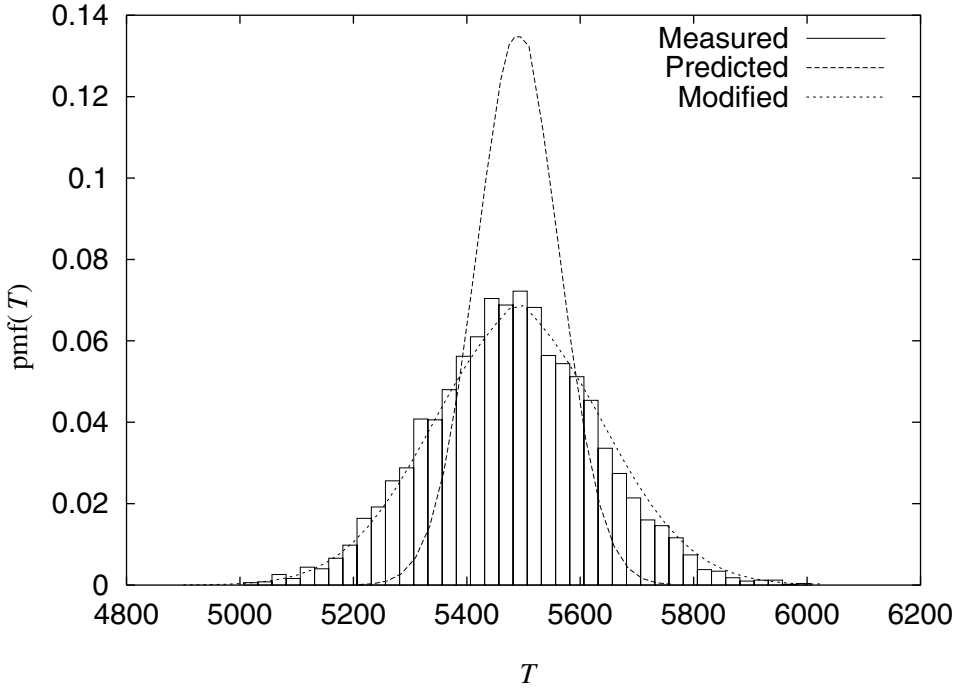

Figure 3.8: The pmf(T) of Straight Selection Sort for $N = 1,000$

To obtain a better prediction for the execution frequency, we modified our prediction by taking covariance into account. The fact is that in general, the variance of a sum of

random variables is not equal to the sum of the individual variances. From Eq. (3.26) it follows that $\mathsf{Var}[T]$ is given by

$$\mathsf{Var}[T] = \sum_{i=2}^{N} \mathsf{Var}[T_{\mathtt{Max},i}] + \sum_{\substack{j=2 \\ j \neq k}}^{N} \sum_{k=2}^{N} \mathsf{Cov}[T_{\mathtt{Max},j}, T_{\mathtt{Max},k}]. \tag{3.66}$$

The last term is responsible for the covariance in each iteration. If the $T_{\mathtt{Max},i}$ are independent, the last term is equal to zero. The measured covariance between the jth and kth iteration of the i-loop over the entire range of data sets is shown by Figure 3.9. The results show that the covariance is indeed positive. The covariance is especially large when the distance between j and k is small. Therefore, in our modified prediction we only neglect the covariance if $\mid j - k \mid \geq 10$. By including the covariance in the analysis we obtain a much better prediction for the variance (which is illustrated by the modified execution time in Figure 3.8). In summary, the experiment shows that although our analysis technique applies, branches should be modeled with caution, because the "loop-carried" covariance cannot always be ignored.

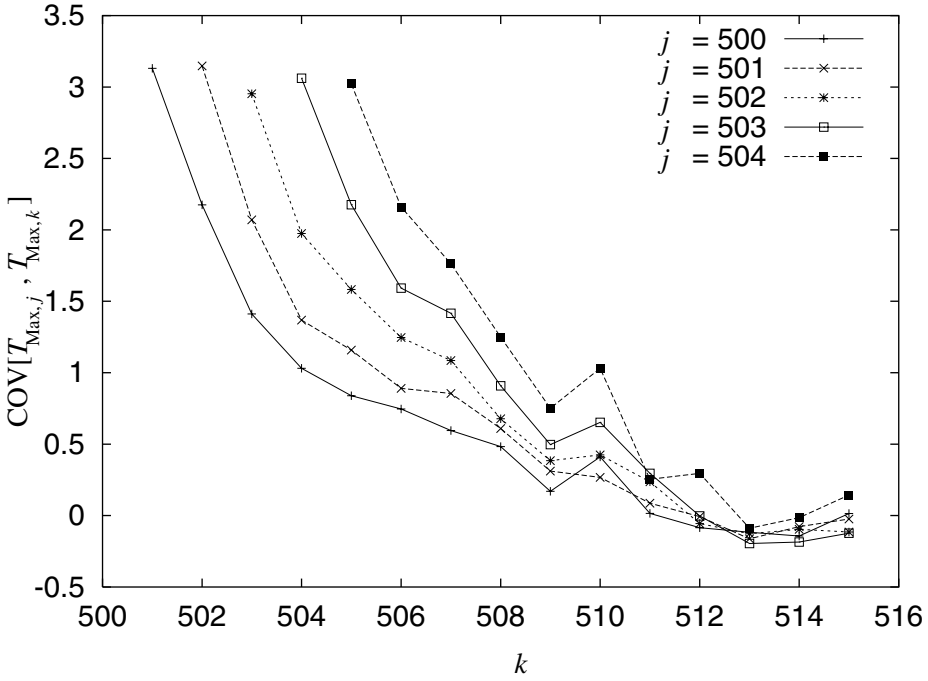

Figure 3.9: The covariance between i-loop iterations $j \geq 500$ and $k \geq 500$

3.5.3 Memory Hierarchy

Thus far, we have analyzed the stochastic effects at the program level, showing that the effect of variance on the branch truth probability and loop bounds is very important. Another source of stochastic behavior is the effect of memory hierarchy, which manifests itself in *basic blocks* having a stochastic execution time. This adds a third source of stochastic behavior of program execution time next to branches and loop bounds.

In this section we apply our analysis technique to predict cache performance. For the purpose of the example, the cache model we use is as follows:

```
if (hit(a))
    move_local(a);
else
    move_global(a);
```

where `a` denotes a memory address. If an address is found in the cache, `hit(a)` is true, and the corresponding data is moved locally to the processor. Otherwise, the data must be moved globally from the lower-level memory hierarchy. In the model we ignore the fact that usually a complete cache *line* consisting of multiple words is loaded from memory.

As driver program we use Quicksort [81], which generates a highly data-set-dependent memory-address stream using *trace-driven simulation* [99]. A trace is usually gathered by interpretively executing a program and recording every main memory location referenced by the program during its execution. We mention two advantages of the trace-driven simulation above over direct measurement:

1. It is possible and easy to vary the parameters of the simulation model, for example cache size, line size, set associativity, replacement algorithm etc. In direct measurement, this is not possible since it requires access to a computer and to hardware measurement tools.

2. The experiment using trace-driven simulation is reproducable, while in direct measurement, we require a single-user machine. This requirement is important because the effectiveness of a cache memory depends on the total workload of the computer system [100].

We have modified Quicksort such that memory reference can be recorded. The Quicksort program contains 27 loads and stores and the instrumented version is given in Appendix E. Note that we interpret loads and stores as memory references. We applied the cache model in all parts of the program where memory reference occurs. Apart from the cache model we account for 6 loop bounds and 8 if statements. As input for Quicksort we have chosen an array with length $1,000$ and the addresses were assigned to element addresses 0 to 999. The number of memory references in the simulation was approximately $N \approx 14,000$. Figure 3.10 shows the resulting trace obtained from the Quicksort program. The simulations show that the memory reference pattern of Quicksort is stochastic, as it differs per data set.

In order to maximize the variance for this particular application, we have chosen a 2-way set associative cache with cache size 0.125K bytes, block size 32 bytes with random replacement, which yields a hit ratio that is sufficiently less than one. For a hit we charged $\tau_l = 10^{-9}$ time units local access time including cache directory search, and for a miss we charged $\tau_g = 10^{-8}$ time units of global memory access time including the cache directory update. In practice, the typical local access was indeed in the nano seconds range and the ratio between τ_g and τ_l can varied from 10 to 100.

Figure 3.11 shows the measured execution time of cache program from measurements with 5,000 random data sets. The mean and variance are 3.09×10^{-5} and 6.55×10^{-13},

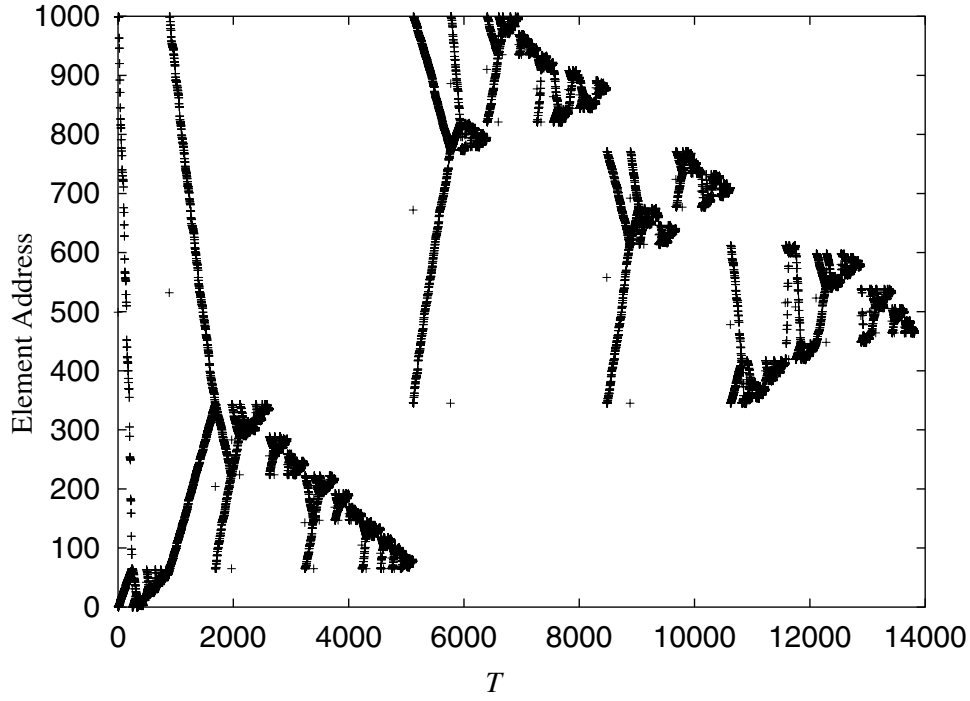

Figure 3.10: Memory references of Quicksort

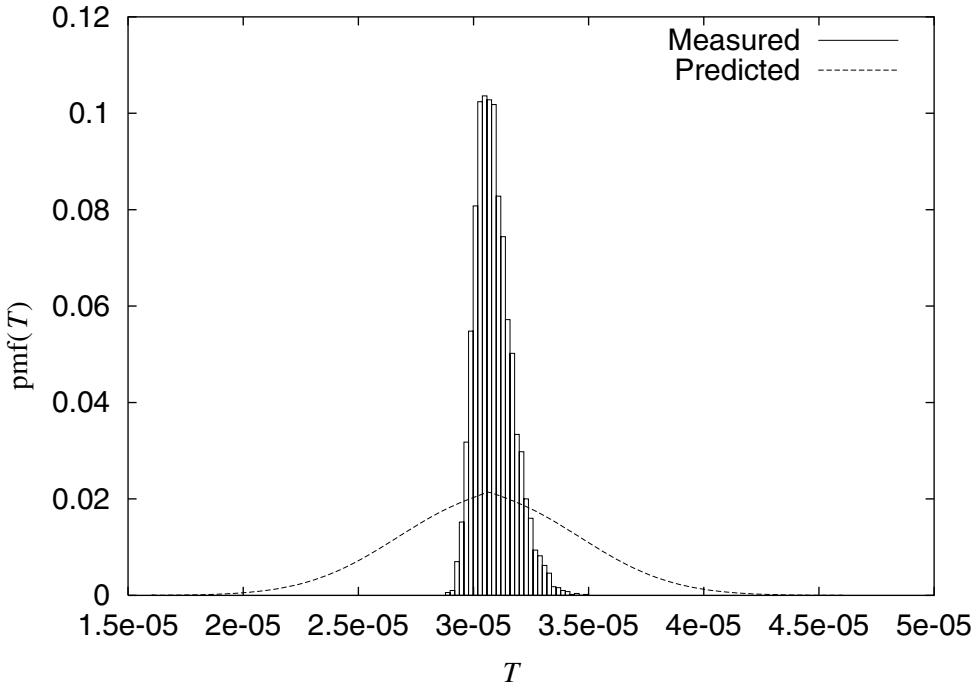

Figure 3.11: The pmf(T) Quicksort

respectively (i.e., a 49% hit rate). The predicted execution time of the cache program is obtained by substituting Eq. (3.53) into Eq. (3.45). Let the branch condition `hit(a)` be modeled by Bernoulli process with parameter $\mathsf{E}[P] = p$, which is determined by instrumenting the if statement to measure the truth probability (i.e., cache hit ratio). The predicted execution time is given in Figure 3.11, where the mean and variance are 3.07×10^{-5} and 1.59×10^{-11}, respectively. In the figure, the mean prediction is accurate while in this case the predicted variance is too large.

To obtain a better prediction for the variance we again include the covariance in our analysis. However, it turns out that including the covariance in the analysis does not improve the predicted variance. Figure 3.12 shows the measured covariance for the dominant outer loop in Quicksort as function of the loop index i. For the modified execution time the mean and variance are 3.07×10^{-5} and 1.58×10^{-11}, respectively. The

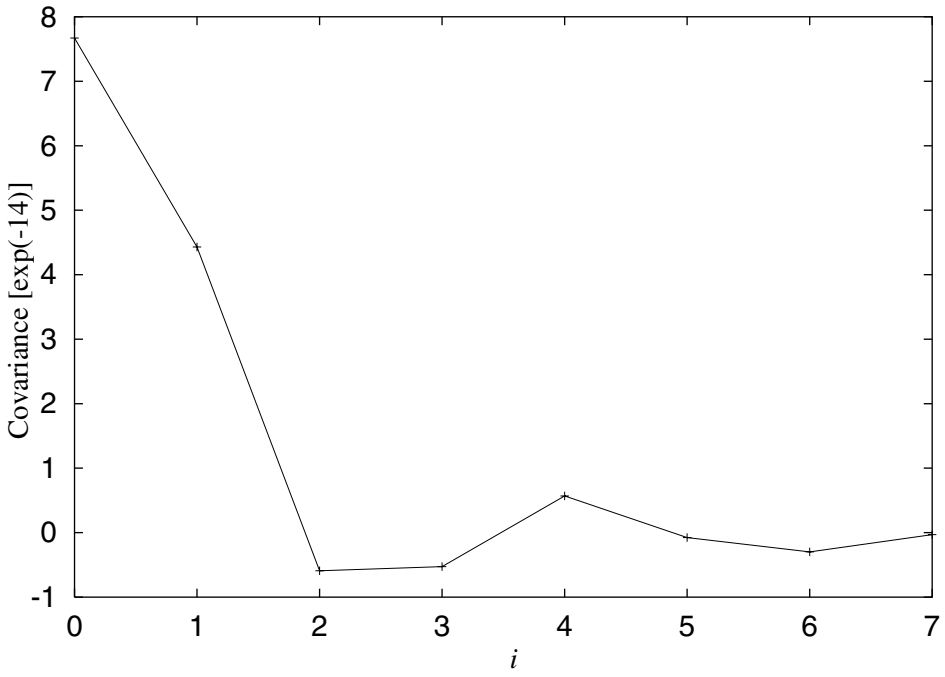

Figure 3.12: The covariance of Quicksort

predicted variance cannot be improved due to the fact that the cache only contributes 0.6% to the total covariance, while the largest part of covariance appears to be contributed by loop bounds and branches in the Quicksort program code.

As program performance is usually largely determined by cache performance, one would also expect the stochastic behavior of the program execution time to depend mostly on cache behavior. Even though in our model the cache is the only source of delay, our results show, however, that for our experiment the cache contribution to the *variance* is small.

3.6 Summary

In this chapter we have introduced our statistical technique which predicts the execution time distribution of data-dependent programs in terms of statistical moments. Using the program control flow structure, our static technique combines the statistical moments of sequential loop bounds, branch probabilities, and basic block execution times in terms of a symbolic process that has a time complexity that is linear in the symbolic size of the program. To illustrate our analysis technique we have applied our analysis to three small sequential test programs, which exhibit a high degree of data dependency. In Vector Scaling and Straight Selection Sort we were able to successfully predict the execution time. In Memory Hierarchy we can predict the mean execution time with high accuracy and the variance with low accuracy. We have also shown that often one cannot make Bernoulli assumptions in modeling branch behavior. The same holds for the assumption that correlations between individual loop iterations often cannot be made. Chapter 4 will further discuss this issue in depth.

We summarize the capability of our analysis technique in Table 3.3 in terms of the prediction accuracy and the appropriate workload. For sequential and conditional compositions, these aspects are exact and general, respectively. Although we have demonstrated the derivation for the first four moments only, higher moments can be also obtained in a similar manner.

For parallel composition, we use an approximate method to represent the first four moments. This approximation is necessary due to the integration problem related to parallel composition. A more elaborate and precise method for obtaining the first four moments in parallel composition is treated in Chapter 5. Note that for all cases, the prediction cost is $O(1)$ and independent of N.

Table 3.3: Summary of moment analysis characteristics

Aspect	Composition		
	Sequential	Conditional	Parallel
Accuracy	exact	exact	approximate
Workloads	general	general	the first four moments

Chapter 4

Conditional Composition

As mentioned in Section 3.5, conditional composition poses additional problems with respect to modeling branch probability. Although the moments calculus underlying conditional composition is well-understood, statistically characterizing branching behavior turns out to be quite far from trivial. Recall Straight Selection Sort in Section 3.5.2. While the prediction accuracy for the mean is very good, that for the variance is quite poor. It shows that a simple Bernoulli model is not always applicable.

As the branches we consider are typically executed in a loop (nest), we will consider the following conditional composition sequence:

```
for (i = 1; i <= N; i++)
    if (C)
        statement_i;
```

as the basic kernel for which we derive the execution time distribution T in terms of the first four moments. Let the truth of branch condition `C` be modeled by the random variable P, while the loop bound `N` and the workload of the ith `statement` instance are represented by random variables N and X_i, respectively. Let the conditional composition have execution time distribution Z. In Section 3.4.2, it is proven that the execution time of the above loop is given by (N-ary sequential composition)

$$\mathsf{E}[T^r] = \mathsf{E}\left[\frac{\mathrm{d}^r}{\mathrm{d}t^r}\left(\sum_{j=0}^{r}\frac{t^j\mathsf{E}[Z^j]}{j!}\right)^N\Bigg|_{t=0}\right], \tag{4.1}$$

where (conditional composition)

$$\mathsf{E}[Z^r] = \mathsf{E}\left[\frac{\mathrm{d}^r}{\mathrm{d}t^r}\left(\sum_{j=0}^{r}\frac{t^j\mathsf{E}[X^j]}{j!}\right)^P\Bigg|_{t=0}\right]. \tag{4.2}$$

Note that $\mathsf{E}[X^r]$ denotes the rth *raw* moment of random variable X, from which the first four moments can be derived straightforwardly. Since we focus on the branch, for simplicity, we let `statement_i` take unit execution time, i.e., $X = 1$. From Section 3.5.2, the first four moments of T are given by

$$\mathsf{E}[T] = \mathsf{E}[N]\mathsf{E}[P], \tag{4.3a}$$

$$\mathsf{Var}[T] = \mathsf{E}[N]\mathsf{Var}[P] + \mathsf{E}[P]^2\mathsf{Var}[N], \tag{4.3b}$$

$$\begin{aligned}\mathsf{Skw}[T] = {} & (\mathsf{E}[N]\mathsf{Skw}[P]\mathsf{Std}[P]^3 + \mathsf{E}[P]^3\mathsf{Skw}[N]\mathsf{Std}[N]^3 \\ & + 3\mathsf{E}[P]\mathsf{Var}[P]\mathsf{Var}[N])/\mathsf{Std}[T]^3,\end{aligned} \tag{4.3c}$$

$$\begin{aligned}\mathsf{Kur}[T] = {} & (\mathsf{E}[N]\mathsf{Var}[P]^2(\mathsf{Kur}[P] - 3) + \mathsf{E}[P]^4\mathsf{Kur}[N]\mathsf{Var}[N]^2 \\ & + 6\mathsf{E}[P]^2\mathsf{Var}[P](\mathsf{Skw}[N]\mathsf{Std}[N]^3 + \mathsf{E}[N]\mathsf{Var}[N]) \\ & + 4\mathsf{E}[P]\mathsf{Var}[N]\mathsf{Skw}[P]\mathsf{Std}[P]^3 + 3\mathsf{Var}[P]^2(\mathsf{E}[N]^2 + \mathsf{Var}[N]))/\mathsf{Std}[T]^4.\end{aligned} \tag{4.3d}$$

Thus, given the first four moments of P and N, we can evaluate the execution time distribution of the above loop using Eq. (4.3).

In our compositional approach we aim to statistically model the behavior of `C`, rather than modeling, e.g., the entire sequential composite, in which case we would sacrifice analytical information on the effect of N and X. However, in statistically modeling the behavior of `C`, of which the truth probability is denoted by P, we are confronted with the fact that, unlike N and X, `C` yields only false (0) or true (1). Being a discrete distribution, the pmf of P can be given, for example, by a histogram where the mass is either in 0 or 1. This interpretation is far from the real behavior of the branch since this interpretation would imply a Bernoulli model. In this interpretation we implicitly assume that the occurrence of 0 and 1 is independent. As shown later on, this model often fails to predict branch behavior that is typically far from memoryless.

In this chapter we address the following question: How can branches be statistically modeled such that the model parameters can be accurately determined by measurements, and can subsequently be successfully used within our compositional approach to obtain execution time distribution prediction?

We propose and evaluate three different approaches to statistically modeling data-dependent branching behavior to reflect the large impact of data-dependent branching on program execution time distribution, as well as the difficulty of integrating branching into a statistical framework. The approaches are called the Empirical approach, the Bernoulli approach, and the ARP (Alternating Renewal Processes) approach, respectively. The Empirical approach was inspired by taking into account that p actually has a *distribution* of its own when measured across the spectrum of input data sets, which allows P to be (indirectly) *measured* through p. In the Bernoulli approach, a branch is simply modeled by a Bernoulli model with (measured) parameter p. In the ARP approach, a branch is modeled in terms of Alternating Renewal Processes; this approach extends the Bernoulli approach by modeling a branch in terms of (measured) two parameters, instead of one. To the best of our knowledge, such an in-depth study on statistical branch modeling in the context of static performance prediction has not been conducted before.

The remainder of the chapter is organized as follows. In Section 4.1, we review current approaches towards modeling branching behavior. In Sections 4.2, 4.3, and 4.4 we describe the Empirical approach, Bernoulli approach, and ARP approach, respectively. In Sections 4.5, 4.6, and 4.7, we test our method using synthetic, Markovian branch distributions, as well as distributions measured from real applications, respectively. Finally, in Section 4.8 we draw our conclusions.

4.1 Branch Modeling

Because data-dependent branching has such a large impact on program behavior, quite a lot of work on branch behavioral modeling has been done, although from an entirely different perspective. Branch modeling is applied for various reasons. In *dynamic* (run-time) approaches, branching behavior is modeled in order to predict the branch outcomes such that performance loss due to, e.g., pipeline stalls can be minimized. This approach is often referred to as *branch prediction*. The branch outcome can be predicted at run time based on the execution history during previous n outcomes [101, 23, 75] or it can be predicted statically at compile time based on either branch profile information [113] or heuristically [10, 17] (e.g., prediction is based on the location of the branch). Although the dynamic approach differs from ours, the branch behavior models it uses are a valuable source of modeling information, as can be seen in Section 4.6.

In *static* approaches, like our statistical performance prediction approach, branching behavior is predicted at compile time with the intention to predict program execution time rather than branch outcomes at run time. Typically, branches are only modeled in terms of their mean truth probability. Approaches to this effect include the work of Adve and Vernon [2], Lester [59], Van Gemund [32], Sarkar [91], and Wagner, Maveric, Graham and Harrison [109]. In these approaches, branch probabilities are effectively assumed to have zero variance, while, in fact, the variance across the spectrum of data sets can be considerable. In the approach by Trivedi [106], variance *is* taken into account as it models branch probabilities in terms of a Bernoulli distribution. In our work we essentially generalize over all the above static approaches.

4.2 Empirical Approach

As an intuitive baseline, in this section we present an empirical approach to modeling branching that is inspired by how branches are measured in practice. Given input data sets, we can study branch patterns using the basic loop kernel mentioned earlier. For instance, consider the branch outcome patterns in Table 4.1, expressed in terms of true (`t`) and false (`f`). For simplicity, the table shows two streams of the same length, i.e.,

Table 4.1: Branch outcome patterns in terms of true and false

Data sets	Branch outcome patterns	N	T
1	`ttfftfftttfffttttftttfffttttftf`	30	16
2	`tttttttfffffffftttttttfffffff`	30	14

$N = 30$, although typically N is stochastic (data dependent). In the first stream shown in the table, there is much more switching between `t` and `f` than in the second stream.

An alternating process whose properties are represented in a similar way as in our branching process in Table 4.1 is that presented by Pham-Gia and Turkkan [79]: it uses two states (an 'on' and an 'off' state) for this purpose. They introduced the ratio between

the proportion of time that the system is on and the total time. They showed that this ratio is a good measure to predict the system behavior for a few cycles ahead. Therefore, to obtain the branch probability, we use random the variable P_m defined by the ratio between the loop execution time T and the loop bound N according to

$$P_m = \frac{T}{N}. \tag{4.4}$$

Similar to [79], for simplicity, we will consider Eq. (4.4) for deterministic (i.e., data-independent) loop bounds N, since deriving the moments of P_m for stochastic N is mathematically very complex [79]. The first four of P_m are given by

$$\begin{aligned}
\mathsf{E}[P_m] &= \mathsf{E}[T/N] = \mathsf{E}[T]/\mathsf{E}[N] = \mathsf{E}[T]/N, && (4.5a)\\
\mathsf{Var}[P_m] &= \mathsf{E}[(T/N)^2] - \mathsf{E}[T/N]^2 \\
&= (\mathsf{E}[T^2] - \mathsf{E}[T]^2)/\mathsf{E}[N]^2 \\
&= \mathsf{Var}[T]/\mathsf{E}[N]^2 = \mathsf{Var}[T]/N^2, && (4.5b)\\
\mathsf{Skw}[P_m] &= (\mathsf{E}[(T/N)^3] - 3\mathsf{E}[T/N]\mathsf{E}[(T/N)^2] + 2\mathsf{E}[T/N]^3)/\mathsf{Var}[P_m]^{3/2} \\
&= (\mathsf{E}[T^3] - 3\mathsf{E}[T]\mathsf{E}[T^2] + 2\mathsf{E}[T]^3)/((\mathsf{Var}[T]/N^2)^{3/2}\mathsf{E}[N]^3) \\
&= (\mathsf{E}[T^3] - 3\mathsf{E}[T]\mathsf{E}[T^2] + 2\mathsf{E}[T]^3)/\mathsf{Var}[T]^{3/2} \\
&= \mathsf{Skw}[T], && (4.5c)\\
\mathsf{Kur}[P_m] &= (\mathsf{E}[(T/N)^4] - 4\mathsf{E}[T/N]\mathsf{E}[(T/N)^3] + 6\mathsf{E}[T/N]^2\mathsf{E}[(T/N)^2] - 3\mathsf{E}[T/N]^4)/\mathsf{Var}[P_m]^2 \\
&= (\mathsf{E}[T^4] - 4\mathsf{E}[T]\mathsf{E}[T^3] + 6\mathsf{E}[T]^2\mathsf{E}[T^2] - 3\mathsf{E}[T]^4)/((\mathsf{Var}[T]/N^2)^2\mathsf{E}[N]^4) \\
&= (\mathsf{E}[T^4] - 4\mathsf{E}[T]\mathsf{E}[T^3] + 6\mathsf{E}[T]^2\mathsf{E}[T^2] - 3\mathsf{E}[T]^4)/\mathsf{Var}[T]^2 \\
&= \mathsf{Kur}[T]. && (4.5d)
\end{aligned}$$

The above results express the moments of P_m in terms of the measured moments of N and T. Conversely, we can, of course, simply reproduce the moments of T from the moments of N and P_m. However, since P_m is derived in terms of a *loop* composition rather than in terms of an isolated branch, P_m is not directly applicable to our compositional method in Eq. (4.2). Yet P_m can serve as the basis for our compositional method. As an estimation of P, we introduce a truth probability P_e, the 'e' denoting our *empirical* approach, by relating Eq. (4.5) with Eq. (4.3) for $\mathsf{Var}[N] = 0$. Then the moments of P_e are defined according to

$$\begin{aligned}
\mathsf{E}[P_e] &= \mathsf{E}[T]/N, && (4.6a)\\
\mathsf{Var}[P_e] &= \mathsf{Var}[T]/N, && (4.6b)\\
\mathsf{Skw}[P_e] &= (\mathsf{Skw}[T]\mathsf{Std}[T]^3)/(N\mathsf{Std}[P_e]^3) \\
&= (\mathsf{Skw}[T]\mathsf{Std}[T]^3)/(N(\mathsf{Var}[T]/N)^{3/2}), \\
&= \mathsf{Skw}[T]/N^{1/2}, && (4.6c)\\
\mathsf{Kur}[P_e] &= ((\mathsf{Kur}[T]\mathsf{Std}[T]^4 - 3N^2\mathsf{Var}[P_e]^2)/(N\mathsf{Var}[P_e]^2)) + 3
\end{aligned}$$

$$= ((\mathsf{Kur}[T]\mathsf{Std}[T]^4 - 3N^2(\mathsf{Var}[T]/N)^2)/(N(\mathsf{Var}[T]/N)^2)) + 3$$
$$= (\mathsf{Kur}[T] - 3)/N + 3. \tag{4.6d}$$

Using Eq. (4.6), we can determine P_e in terms of N and T. To obtain the first four moments of T using the Empirical approach, denoted by T_e, we substitute P_e into Eq. (4.3).

Clearly, the Empirical approach delivers high accuracy, provided that N is deterministic. However, although the moments of P_e have been derived empirically, the branching behavior itself has not been modeled explicitly (i.e., we have applied a black-box, rather than a white-box approach). Consequently, we have no insight in the effect of N and X, the opposite of what we want from our compositional method (once the isolated branch has been modeled, we still want to be able to vary N and X.). Hence, in the next sections we study the Bernoulli and ARP approach. The Empirical approach, however, serves as a reference model, with which the alternative approaches will be compared.

4.3 Bernoulli Approach

An intuitive approach to modeling branches statistically is to assume P to be a Bernoulli trial with a *deterministic* parameter p that equals the average value of the truth frequency ratio as profiled across an input data set training corpus. The Bernoulli random variable P_b, which will serve as an estimate of P, is defined by

$$P_b = \begin{cases} 1, & \text{if } \mathtt{C} = \text{true}, \\ 0, & \text{if } \mathtt{C} = \text{false}. \end{cases} \tag{4.7}$$

In other words, the random variable P_b maps true and false branch outcomes to ones and zeros, respectively. For instance, consider Table 4.2, which shows the branch outcome patterns corresponding to those in Table 4.1, expressed in terms of ones and zeros. Table 4.2

Table 4.2: Branch outcome patterns in terms of one and zero

Data sets	Branch outcome patterns	p
1	11001001100011110110001111001 0	0.5
2	11111110000000011111110000000 0	

also shows that $p = 0.5$ as determined from the entire measurement (i.e., when all rows are combined).

For a Bernoulli variable with parameter p, the rth raw moment of P_b is given by

$$\mathsf{E}[P_b^r] = p. \tag{4.8}$$

From Eq. (4.8), the first four moments are given by

$$\mathsf{E}[P_b] = p, \tag{4.9a}$$
$$\mathsf{Var}[P_b] = p(1-p), \tag{4.9b}$$
$$\mathsf{Skw}[P_b] = p(1-p)(1-2p)/\mathsf{Std}[P_b]^3, \tag{4.9c}$$
$$\mathsf{Kur}[P_b] = p(1-p)(1-3p+3p^2)/\mathsf{Std}[P_b]^4. \tag{4.9d}$$

Eq. (4.9) shows that the moments of P_b can be determined from the single value p. Taking P_b as estimator for P in Eq. (4.7) we obtain the first four moments of T, denoted by T_b.

By definition, a Bernoulli model is only appropriate for branches that do not depend on previous outcomes, while many practical branches are far from memoryless. For instance, consider a cyclic branch which generates $1010\ldots1010$ across the training data sets, i.e., $p = 0.5$. While obviously $\mathsf{Var}[T] = 0$ for N constant, in contrast Eq. (4.9b) predicts $\mathsf{Var}[T_b] = 0.25\mathsf{E}[N]$ (linear in $\mathsf{E}[N]$). Consequently, the Bernoulli model is often incapable to provide a statistical explanation of a program execution time variance (and higher moments) as caused by many branches in practice. Rather than a branching model using a single value p, we use an alternative one based on *two* parameters that we present in the next section.

4.4 ARP Approach

In reality, the variance of T_m is sensitive to permutation between true and false values (consider all 1's for data set 1, and all 0's for data set 2). In contrast, in the Bernoulli approach the measurement of p is insensitive to any permutation. In this section, we introduce the use of Alternating Renewal Process (ARP) theory since ARP takes the permutation into account.

The ARP theory has been used in various applications relating to the branching processes, such as estimating cache miss ratios [112], predicting the ratio of opening/closing rates of biological membranes due to fluctuation [16], determining the instantaneous reliability in modeling aircraft systems [89], and modeling air-conditioning loads (the ON duration and OFF duration) on electrical power systems [68]. These good results motivated us to investigate the use of ARP in statistical modeling of branching to predict static performance.

While the Bernoulli process considers the branch outcomes individually, ARP considers the outcomes as consecutive trues (in terms of up time, U) or falses (in terms of down time, D) from which the total execution time distribution T can be derived. For instance, consider Table 4.3, which shows the branch outcome patterns corresponding to those in Table 4.1, expressed in terms of U and D. Each cluster of true and false outcomes is

Table 4.3: Branch outcome patterns in terms of U and D

Data sets	Branch outcome patterns	E	Var
1	$U \rightarrow$ 2 1 2 4 2 4 1 $D \rightarrow$ 2 2 3 1 3 2 1	[U]=3.3	[U]=4.9
2	$U \rightarrow$ 7 7 $D \rightarrow$ 8 8	[D]=3.3	[D]=6.7

labeled according to the cluster length and accounted for in terms of the sample of U and D, respectively, each of which comprises nine samples as shown in the table. By combining the samples across all runs, we can determine the mean and variance of U and D, and evaluate the branch probability as follows.

To obtain the branch probability P we use the estimator P_a based on ARP. Since in ARP it is customary to address the analysis in terms of total down time, T_D, rather than up time, T_U, in the following we will derive the moment of $Q_a = 1 - P_a$. Note that T_U is equal to the execution time distribution T.

Following [70], we introduce a random variable which is defined as the conditional probability of $T_D(t)/t$, t denoting the time, given that $T_D(t) > 0$ according to

$$Q_a(t,x) = \mathsf{P}\left[\frac{T_D(t)}{t} \leq x \mid T_D(t) > 0\right]. \tag{4.10}$$

Note that t is comparable with N in the conditional composition sequence. In the rest of this section we simply use Q_a rather than $Q_a(t,x)$. The reason for representing Q_a using Eq. (4.10) is twofold. First, Eq. (4.10) is a valid distribution for branching model since Eq. (4.10) lies in the interval $[0,1]$. Second, Eq. (4.10) represents known properties of $T_D(t)$ from which the exact mean and variance of $T_D(t)$ can easily be derived [70].

In [70] it is shown that the density of Q_a can be approximated using the beta distribution according to

$$f_{Q_a}(x) \approx \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} x^{a-1}(1-x)^{b-1} \quad \text{for } 0 < x < 1, a > 0, b > 0, \tag{4.11}$$

where

$$a = \frac{(\mathsf{E}[D]^2(\mathsf{E}[U] - \mathsf{Var}[U]) + \mathsf{E}[U]^2(\mathsf{E}[D] - \mathsf{Var}[D]))\mathsf{E}[D]}{(\mathsf{E}[D] + \mathsf{E}[U])(\mathsf{E}[D]^2\mathsf{Var}[U] + \mathsf{E}[U]^2\mathsf{Var}[D])}, \quad b = \frac{\mathsf{E}[U]}{\mathsf{E}[D]}\, a. \tag{4.12}$$

Note that the beta distribution exhibits a large diversity of shapes in a finite interval so that the approximation does not limit our analysis, as Q_a is by definition finite in the interval $[0,1]$. The rth raw moment of Q_a is given by

$$\mathsf{E}[Q_a^r] = \frac{\Gamma(a+b)\Gamma(a+r)}{\Gamma(a)\Gamma(a+b+r)}. \tag{4.13}$$

From Eqs. (4.12) and (4.13) the mean and variance of Q_a can be expressed in terms of input moments D and U according to

$$\mathsf{E}[Q_a] = \frac{\mathsf{E}[D]}{\mathsf{E}[D] + \mathsf{E}[U]}, \tag{4.14a}$$

$$\mathsf{Var}[Q_a] = \frac{\mathsf{E}[D]^2\mathsf{Var}[U] + \mathsf{E}[U]^2\mathsf{Var}[D]}{(\mathsf{E}[D] + \mathsf{E}[U])^3}. \tag{4.14b}$$

Extending this result derived in [70] for the purpose of our moment approach we have derived the skewness and kurtosis of Q_a in Appendix C.2.2 according to

$$\mathsf{Skw}[Q_a] = \frac{-2(\mathsf{E}[D] - \mathsf{E}[U])(\mathsf{E}[D] + \mathsf{E}[U])^2\mathsf{Std}[Q_a]}{\mathsf{E}[D]^2(\mathsf{E}[U] + \mathsf{Var}[U]) + \mathsf{E}[U]^2(\mathsf{E}[D] + \mathsf{Var}[D])}, \tag{4.14c}$$

$$\begin{aligned}\mathsf{Kur}[Q_a] = {} & 3(\mathsf{E}[D] + \mathsf{E}[U])[(\mathsf{E}[D] + \mathsf{E}[U])\mathsf{E}[D]^2\mathsf{E}[U]^2 \\ & + (2(\mathsf{E}[D] - \mathsf{E}[U])^2 + \mathsf{E}[D]\mathsf{E}[U])(\mathsf{E}[D]^2\mathsf{Var}[U] + \mathsf{E}[U]^2\mathsf{Var}[D])] \\ & \times [\mathsf{E}[D]^2(\mathsf{E}[U] + 2\mathsf{Var}[U]) + \mathsf{E}[U]^2(\mathsf{E}[D] + 2\mathsf{Var}[D])]^{-1} \\ & \times [\mathsf{E}[D]^2(\mathsf{E}[U] + \mathsf{Var}[U]) + \mathsf{E}[U]^2(\mathsf{E}[D] + \mathsf{Var}[D])]^{-1}.\end{aligned} \tag{4.14d}$$

Note that the random variables P and Q are correlated linearly, i.e., $P = 1 - Q$. Consequently,

$$\mathsf{E}[P] = 1 - \mathsf{E}[Q],\ \mathsf{Var}[P] = \mathsf{Var}[Q],\ \mathsf{Skw}[P] = -\mathsf{Skw}[Q] \text{ and } \mathsf{Kur}[P] = \mathsf{Kur}[Q]. \quad (4.15)$$

Using Eqs. (4.14) and (4.15), we can express the first four moments of P_a in terms of the input moments $\mathsf{E}[U], \mathsf{Var}[U], \mathsf{E}[D]$, and $\mathsf{Var}[D]$. Then by substituting P_a into Eq. (4.3) we obtain the first four moments of T, denoted by T_a.

Since the ARP approach takes into account clustering effects in branch outcome patterns, the approach applies to semi-Markovian branches [88]. Since the up and down time (U and D) may take any valid distribution, the semi-Markovian branches is a superset of those having Bernoulli behavior. We have proven in Appendix C.2.2 that if U and D are geometrically distributed, then the ARP approach reduces to the Bernoulli approach.

4.5 Synthetic Workloads

In this section we evaluate the three approaches by predicting the execution time distribution of the earlier loop kernel for different standard distributions for P. To exclude other potential sources of error, we choose N to be deterministic while `statement` takes unit execution time, i.e., $X = 1$. Since the mean of T for all approaches is exact, we need only consider the higher moments of T. In our experiments we exclusively focused on the execution time variance, which next to the mean is the most important moment, especially in the prediction of parallel composition [1]. The variance error is defined as follows

$$\varepsilon = \frac{\mid \mathsf{Var}[T_m] - \mathsf{Var}[T_p] \mid}{\mathsf{Var}[T_m]}, \quad (4.16)$$

where T_m and T_p are values obtained from measurement and prediction, respectively. Since T_m is the number of taken branches in a run, it can be obtained straightforwardly (cf. Tables 4.1, 4.2, and 4.3). In some cases measurement is not necessary since T_m can be derived analytically. In our experiments, we predicted the execution time using Eq. (4.3b), where the three models for P are P_e (the Empirical approach, Eq. (4.6)), P_b (the Bernoulli approach, Eq. (4.9)), and P_a (the ARP approach, Eq. (4.14)).

4.5.1 Bernoulli Branches

In our first experiment we chose a genuine Bernoulli branch, i.e., $P = P_b$, to test whether all three approaches yield the same results. It is obvious that $P_e = P_b$, while P_a is obtained as follows. According to Bernoulli behavior, let D and U be *geometrically* distributed with parameter p such that $\mathsf{E}[U] = 1/(1-p)$ and $\mathsf{Var}[U] = p/(1-p)^2$ and $\mathsf{E}[D] = 1/p$ and $\mathsf{Var}[D] = (1-p)/p^2$. From Eqs. (4.14) and (4.15), it follows that the moments of P_a in Eq. (4.14) conform to those of P_b in Eq. (4.9) as derived in Appendix C.2.2. It holds that

$$\mathsf{E}[P_a] = p, \quad (4.17a)$$

$$\mathsf{Var}[P_a] = p(1-p), \quad (4.17b)$$

$$\mathsf{Skw}[P_a] = p(1-p)(1-2p)/(p(1-p))^{3/2} \quad (4.17c)$$

$$\mathsf{Kur}[P_a] = p(1-p)(1-3p+3p^2)/(p(1-p))^2 \quad (4.17d)$$

Thus, the P's are equal in terms of their moments ($\mathsf{E}[P^r] = \mathsf{E}[P_e^r] = \mathsf{E}[P_b^r] = \mathsf{E}[P_a^r]$). The moments agree *exactly* with those obtained with the ARP approach, even though the latter is an approximation method due to the use of the beta distribution.

4.5.2 Deterministic Branches

In the second experiment, we apply the approaches to *deterministic* branches, i.e., some predetermined number of true and false evaluations (e.g., cyclic branches). Clearly, for deterministic branches, the variance of execution time is equal to zero ($\mathsf{Var}[T_m] = 0$). To generate the branches let D and U be deterministic according to $\mathsf{E}[D] = d, \mathsf{Var}[D] = 0$ and $\mathsf{E}[U] = u, \mathsf{Var}[U] = 0$. For P_e in the Empirical approach, Eq. (4.6) yields

$$\mathsf{E}[P_e] = \frac{u}{d+u}, \quad \mathsf{Var}[P_e] = 0,\ \mathsf{Skw}[P_e] = 0 \text{ and } \mathsf{Kur}[P_e] = 3 \tag{4.18}$$

since in Eq. (4.5) $P_m = u/d$ is constant. For the ARP approach, the same (correct) moment values are obtained from Eq. (4.14). By substituting P_e and P_a into Eq. (4.3), we obtain $\mathsf{Var}[T_e] = \mathsf{Var}[T_a] = 0$. For the Bernoulli approach it follows that

$$p = \frac{u}{d+u}. \tag{4.19}$$

The prediction based on the Bernoulli estimator P_b results in a dramatic maximum variance error of $\varepsilon \to \infty$ ($\mathsf{Var}[T_m] = 0$ in Eq. (4.16)) for $d = u$, i.e., $p = 1/2$ and yields a minimum error of $\varepsilon \to 0$ as $p \to 0$ or $p \to 1$.

4.5.3 Uniform Branches

As a final experiment using synthetic workloads, we let D and U take discrete *uniform* distributions with sample space $[a_D, b_D]$ and $[a_U, b_U]$, respectively, where $1 \leq a_X \leq b_X$. The mean and variance of D and U are given by $\mathsf{E}[X] = (a_X + b_X)/2$ and $\mathsf{Var}[X] = (b_X - a_X)^2/12$. We choose $a_D = a_U = 1$ while we vary b_D and b_U. To obtain T_m, we simulate the branching process for $N = 50,000$ over $6,000$ random data sets. For the Empirical approach, T_e is equal to T_m ($\varepsilon = 0$) since N is constant. For the Bernoulli approach, we obtain $p = (1 + b_U)(2 + b_D + b_U)$. Subsequently we applied p in Eqs. (4.9) and (4.3) to obtain the variance error of the Bernoulli approach for various values of b_D and b_U as shown in Figure 4.1. The figure shows that ε is extremely large for small b_D and b_U, while slowly decreasing to zero for increasing b_D and b_U. For $b_D = b_U = 1$ it holds that $\varepsilon \to \infty$ since P is deterministic, as discussed previously. Clearly, for these branches that are *not* memoryless, the Bernoulli model is useless.

In contrast to the Bernoulli approach, the ARP approach has a negligible error, as shown by Figure 4.2. For all values of b_D and b_U, the error is consistently below 1%, which is within the noise margin. Note that the ARP approach is clearly applicable to all workloads.

Summarizing this section, we conclude that all approaches are correct for branches with the memoryless Bernoulli workload. When the workload is no longer memoryless, only the Empirical and ARP approach can be applied. The results show that both approaches consistently perform equal to or better than the Bernoulli approach, effectively reducing the variance error by more than one order of magnitude.

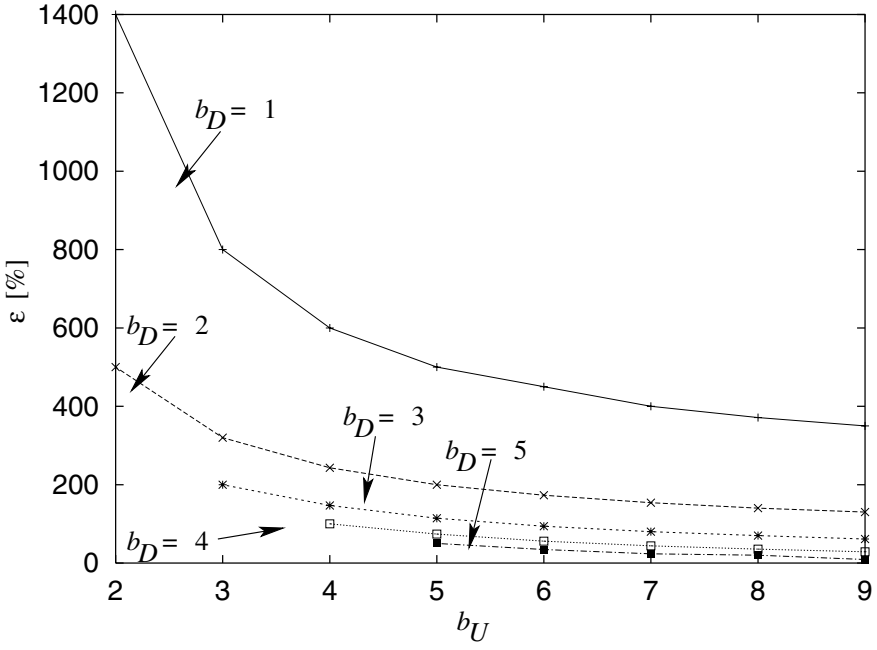

Figure 4.1: ε [%] for the Bernoulli method

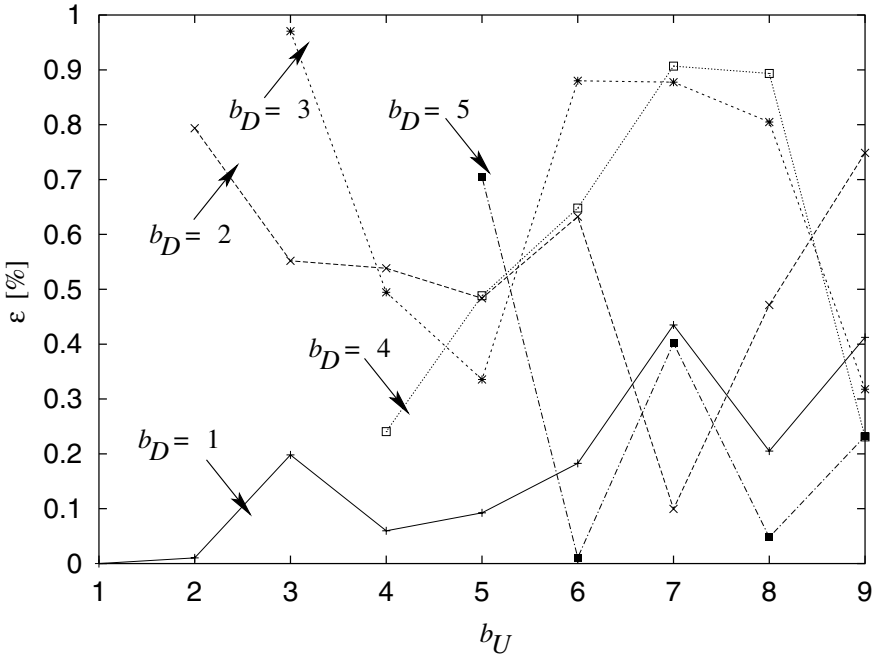

Figure 4.2: ε [%] for the ARP method

4.6 Markovian Workloads

In this section we evaluate the three approaches on branching streams generated from state diagrams adopted from [115] and shown in Figure 4.3, where the branching behavior is not memoryless. Originally these state diagrams were Moore finite-state machines (FSM) used to predict dynamic branch behavior. The current state determines whether the next branch is predicted to be taken and the subsequent state transition depends on whether the branch was actually taken. Designed to achieve high prediction performance, the FSMs represent a model of realistic branching behavior, which make them useful as a basis for synthesizing branches for our study. In fact, these models have been successfully proven to resemble real program branches [75, 101, 113, 115]. To enable the use of the state diagrams in our experiments we modified the FSMs such that streams of ones and zeros can be generated, according to the following Markov interpretation shown by Figure 4.3. Each state is labeled by ({a,b,c,d},{0,1}). The output stream is determined by the state

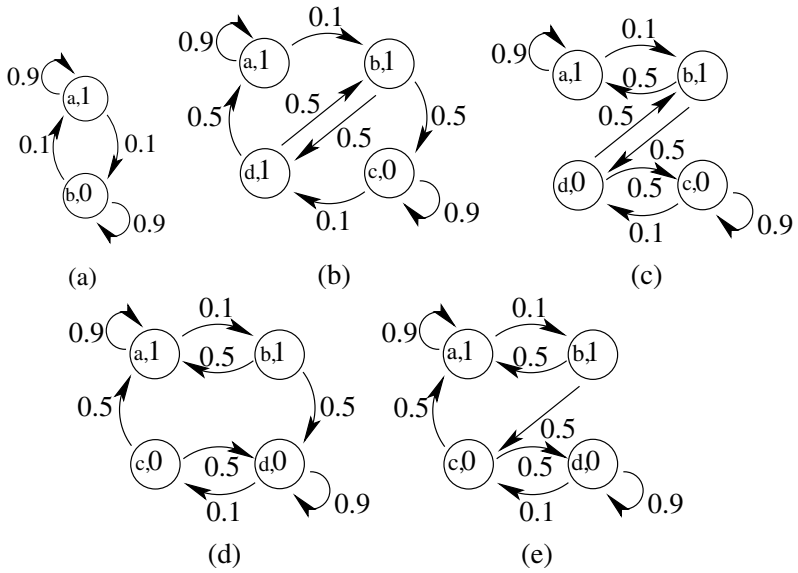

Figure 4.3: Markov chains of branch trace generator.

label {0,1}, while the transition between two states is determined by taking a Bernoulli sample with parameter shown by the value of the corresponding arc. Our experiments have indeed shown that when supplied with a branch stream generated by our generators, the original branch predictors achieve hit rates ranging from 90% to 96%, similar to the ones reported in [115]. This proves the validity of our interpretation. (An identical branch behavior could also have been achieved by a different interpretation where the truth probability is associated with the states rather than the arcs.) Note that the arc values were chosen such that all nodes will be visited with probability larger than 0.

As is to be expected, the variance error of the Empirical approach is equal to zero, while that of the other two approaches is shown in Figure 4.4. For the Bernoulli approach, the variance error is approximately between 90% and 100%, showing that the Bernoulli method is indeed inadequate to model Markovian branches. In sharp contrast to the

Bernoulli approach, the ARP approach yields exact results. For small N, the error is due to inaccuracies in measuring U and D. Again, the ARP approach is much better than the Bernoulli approach, and has the same accuracy as the Empirical approach in capturing branch behavior.

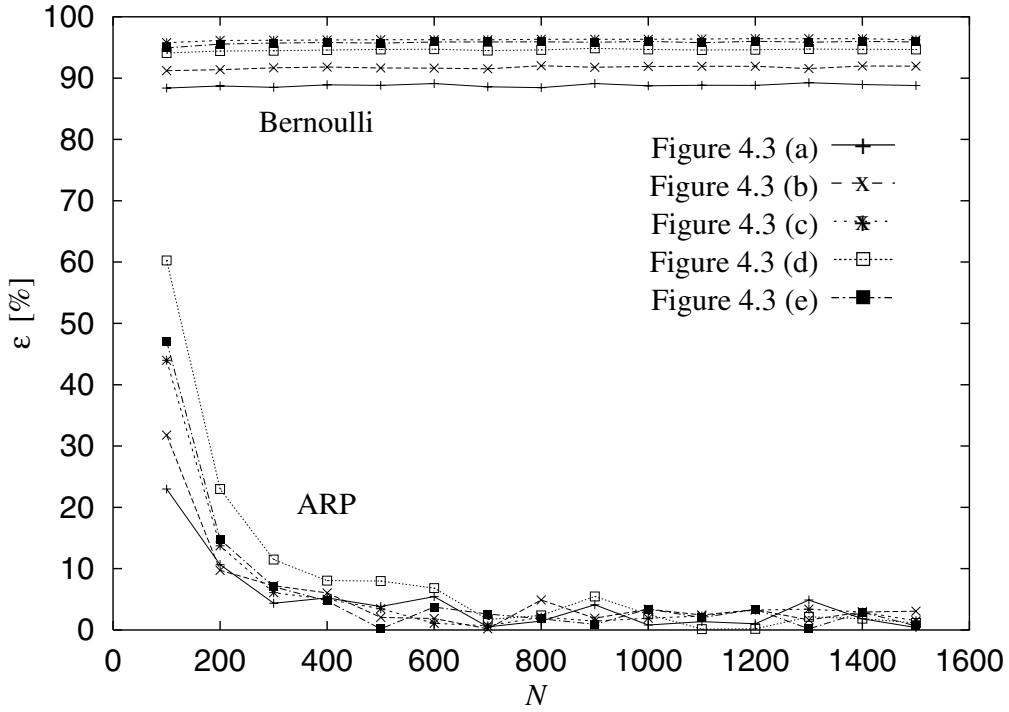

Figure 4.4: Variance error [%] for Markovian workloads

4.7 Empirical Workloads

In this section we illustrate to what extent the branching models apply in practice. We have measured and predicted T for codes containing stochastic branches for 6,000 input data sets, using our counter-based measurement technique. Our experiments include simple kernels, namely Vector Scaling (VS), Straight Selection Sort (SSS) and Cache Simulator (CS) (introduced in Section 3.5.3), as well as larger applications, namely Gaussian Elimination (GE), Single Source Shortest Path (SSSP), Quicksort (iterative fashion), and Compress from SPECint95. Other benchmarks in SPECint95 are not considered in our experiments due to the absence of stochastic branches and/or an insufficient number of input data sets. In all benchmarks we ignored all data-independent branches in our experiments.

In the following we briefly describe the working of the particular benchmarks. Vector Scaling scales a vector of constant length if the vector element is not equal to zero. SSS and Quicksort sort a vector of constant length in increasing order. SSSP finds the shortest path between two nodes for a weighted graph. Gaussian Elimination reduces a

two-dimensional matrix to triangular form. Cache Simulator simulates hit and miss in a cache for obtaining data given cache addresses. Compress is a file compression utility. Specifically, for Compress we only consider branches (b_i) with high invocation frequency.

We summarize our results in Table 4.4 including the error percentages. The highest measured variance $\mathsf{Var}[T_m]$ is 2.0 10^8 in Compress b_2. Significant errors are produced by SSSP b_2, Quicksort b_2 and b_6, and Compress b_4, which are branches that execute a `break` statement, in which case the effective loop frequency is highly stochastic, as well as fully correlated with P. Clearly, to this branch category our conditional composition models cannot be applied. Other highly correlated branches that do not execute `break`, which produce a large error, are SSSP b_1 and b_3, and Compress b_1 and b_2. For this branch category the large error is due to the correlation between branch invocations. However, when the loop frequency is constant, the Empirical approach can be applied, such as in SSS and Quicksort b_1.

The comparison between the prediction error of the various approaches allows us to characterize the branches as Bernoulli (ε in $\mathsf{Var}[T_e], \mathsf{Var}[T_b]$ and $\mathsf{Var}[T_a]$ is small), Markovian (ε in $\mathsf{Var}[T_b]$ is large), and other branches (ε in $\mathsf{Var}[T_b]$ and $\mathsf{Var}[T_a]$ is large). From the 19 branches shown in Table 4.4, 3 branches can be categorized as Markovian branches, i.e., Gaussian Elimination, Quicksort b_5 and Compress b_5. From the other 16 branches, 5 branches can be categorized as Bernoulli branches, i.e., Vector Scaling, Quicksort b_3, b_4 and b_7, and Compress b_3. Thus, our compositional approach applies to almost half of the 19 branches.

Table 4.4: $\mathsf{Var}[T]$ prediction based on P_a, P_b and P_e (ε [%])

Experiment	$\mathsf{Var}[T_m]$	$\mathsf{Var}[T_e]$	$\mathsf{Var}[T_b]$	$\mathsf{Var}[T_a]$	Type
VS	$9.0\,10^2$	$9.0\,10^2$ $(0.0\,10^0)$	$9.0\,10^2$ $(0.0\,10^0)$	$9.0\,10^2$ $(0.0\,10^0)$	B
SSS	$2.2\,10^4$	$2.2\,10^4$ $(0.0\,10^0)$	$5.4\,10^3$ $(7.6\,10^1)$	$1.5\,10^4$ $(3.2\,10^1)$	
CS	$1.0\,10^6$	$7.6\,10^5$ $(2.7\,10^1)$	$7.2\,10^5$ $(2.7\,10^1)$	$1.8\,10^6$ $(7.7\,10^1)$	
GE	$9.4\,10^3$	$9.4\,10^3$ $(5.6\,10^{-1})$	$6.5\,10^3$ $(3.2\,10^1)$	$9.4\,10^3(5.6\,10^{-1})$	M
SSSP b_1	$3.6\,10^4$	$3.1\,10^4$ $(1.4\mathrm{e}\,10^1)$	$7.7\,10^3$ $(7.9\,10^1)$	$2.4\,10^6$ $(6.5\,10^3)$	
SSSP b_2	$4.4\,10^2$	$6.2\,10^3$ $(1.3\,10^3)$	$4.3\,10^3$ $(8.6\,10^2)$	$4.3\,10^5$ $(9.6\,10^4)$	
SSSP b_3	$3.6\,10^4$	$3.1\,10^4$ $(1.4\,10^1)$	$7.7\,10^3$ $(7.9\,10^1)$	$3.1\,10^4$ $(6.5\,10^3)$	
Quicksort b_1	$1.7\,10^1$	$1.7\,10^1$ $(0.0\,10^0)$	$1.1\,10^2$ $(5.7\,10^2)$	$3.2\,10^1$ $(1.9\,10^2)$	
Quicksort b_2	$6.7\,10^1$	$5.3\,10^2$ $(6.8\,10^2)$	$6.9\,10^2$ $(9.2\,10^2)$	$7.0\,10^2$ $(9.4\,10^2)$	
Quicksort b_3	$5.3\,10^1$	$5.2\,10^1$ $(8.2\,10^{-1})$	$5.2\,10^1$ $(1.6\,10^0)$	$5.2\,10^1$ $(1.1\,10^0)$	B
Quicksort b_4	$4.3\,10^1$	$4.3\,10^1$ $(0.0\,10^0)$	$4.3\,10^1$ $(0.0\,10^0)$	$4.3\,10^1$ $(0.0\,10^0)$	B
Quicksort b_5	$4.2\,10^1$	$4.3\,10^1$ $(5.5\,10^{-1})$	$4.4\,10^1$ $(3.5\,10^0)$	$4.3\,10^1$ $(1.5\,10^{-1})$	M
Quicksort b_6	$1.7\,10^1$	$3.1\,10^1$ $(8.5\,10^1)$	$1.8\,10^2$ $(9.6\,10^2)$	$7.5\,10^1$ $(4.5\,10^2)$	
Quicksort b_7	$5.1\,10^1$	$5.2\,10^1$ $(3.3\,10^0)$	$5.2\,10^1$ $(2.3\,10^0)$	$5.2\,10^1$ $(3.1\,10^0)$	B
Compress b_1	$3.9\,10^7$	$2.4\,10^7$ $(3.8\,10^1)$	$2.0\,10^7$ $(4.7\,10^1)$	$2.0\,10^7$ $(4.7\,10^1)$	
Compress b_2	$2.0\,10^8$	$1.8\,10^8$ $(1.1\,10^1)$	$1.7\,10^8$ $(1.3\,10^1)$	$1.7\,10^8$ $(1.4\,10^1)$	
Compress b_3	$1.4\,10^6$	$1.4\,10^6$ $(1.5\,10^{-1})$	$1.4\,10^6$ $(6.9\,10^{-1})$	$1.4\,10^6$ $(7.0\,10^{-1})$	B
Compress b_4	$2.5\,10^1$	$1.4\,10^2$ $(4.3\,10^2)$	$1.7\,10^2$ $(5.7\,10^2)$	$2.1\,10^2$ $(7.3\,10^2)$	
Compress b_5	$9.3\,10^0$	$9.3\,10^0$ $(0.0\,10^0)$	$2.7\,10^0$ $(7.1\,10^1)$	$1.0\,10^1$ $(7.8\,10^0)$	M

4.8 Summary

Data-dependent branches are an important source of program execution time variability across the spectrum of possible input data sets. In this chapter we have evaluated the Empirical, the Bernoulli, and the ARP approach to statistically modeling branching behavior, to be used within our compositional method to predict program execution time distribution in terms of statistical moments.

The summary of the branching model characteristics is given in Table 4.5. Unlike the two other approaches, the Empirical approach is not based on any branching model since it is implicitly assumed that the branch cannot be separated from the loop. Consequently, there is no specific branch behavior for which the approach is fully accurate. Furthermore, the Empirical approach assumes that N is deterministic. Each approach has a specific measurement technique to obtain P. In the Empirical approach, we have to record the sample of P_e strictly per data set, i.e., the data-set axis is taken into account (t.i.c.) (vertical direction in Tables 4.1, 4.2, and 4.3). The sample of P_e is independent from the permutation of zeros and ones in the loop-bound axis (horizontal direction in Tables 4.1, 4.2, and 4.3).

For the Bernoulli approach, the measurement is the simplest since P_b can be obtained from the samples irrespective of the permutation in the data-set axis (vertical) and in the loop-bound axis (horizontal). For the ARP approach, we measure D and U per clusters.

Table 4.5: Summary of branching approach characteristics

Aspects	P_e	P_b	P_a
Branch model	not applicable	Bernoulli	semi Markovian
Loop bound N	deterministic	stochastic	stochastic
Measurement	per data set	whole data set	per cluster
Loop-bound axis	not t.i.c.	not t.i.c.	t.i.c.
Data-set axis	t.i.c.	not t.i.c.	t.i.c.

While the Empirical approach is based on merely measuring branching behavior in terms of the surrounding loop construct, the other approaches aim at deriving a statistical model of the branch itself, which offers a higher level of compositionality. Our measurement results, based on synthetic as well as real programs, show that the Empirical approach delivers the highest accuracy, whereas the alternative approaches trade accuracy for compositionality. For Markovian branches (half of the branches studied), the compositional approaches deliver high prediction accuracy. In contrast to what we might expect intuitively and from our synthetic experiments, in real programs, the two-parameter ARP approach does not always outperform the one-parameter Bernoulli approach. This is because the ARP approach is much more sensitive to correlation than the Bernoulli approach.

Chapter 5

Parallel Composition

As mentioned in Section 3.4.5, expressing $\mathsf{E}[Y^r]$ explicitly in terms of $\mathsf{E}[X^r]$ poses a major problem due to the integration problem in Eq. (3.59). Recall the and-parallel composition:

```
forall (i = 1; i <= N; i++)
    statement(i);
```

which specifies N processes running in parallel without any intermediate form of synchronization. Let `statement(i)` have workload denoted by X_i. The total execution time, denoted Y, is given by

$$Y = \max(X_1, X_2, \ldots, X_N). \tag{5.1}$$

Let X_i be independent from N. The cdf of Y is given by

$$F_Y(x) = \prod_{i=1}^{N} F_{X_i}(x). \tag{5.2}$$

From Eq. 5.2 we derive two important special cases: N-ary parallel composition involving *iid* workloads and binary parallel composition involving *different* workloads. For N-ary parallel composition (X_i are iid), Eq. (5.2) reduces to

$$F_Y(x) = (F_X(x))^N. \tag{5.3}$$

From Eqs. (3.9) and (5.3) we obtain

$$\begin{aligned} \mathsf{E}[Y^r] &= \int_{-\infty}^{\infty} x^r \mathrm{d}F_Y(x) \\ &= \int_{-\infty}^{\infty} x^r \mathrm{d}(F_X(x))^N \\ &= N \int_{-\infty}^{\infty} x^r (F_X(x))^{N-1} \mathrm{d}F_X(x). \end{aligned} \tag{5.4}$$

In general, the right-hand side integration is fundamentally impossible to solve analytically in such a way that $\mathsf{E}[Y^r]$ can be expressed directly as function of $\mathsf{E}[X^r]$.

Unlike N-ary parallel composition, binary parallel composition poses a much more complicated problem since two *different* workloads are involved in the analysis. When X_1 and X_2 are not identical, from Eq. (5.2) the cdf is given by

$$F_Y(x) = F_{X_1}(x) F_{X_2}(x). \tag{5.5}$$

From Eqs. (3.9) and (5.5) we obtain

$$\begin{aligned}\mathsf{E}[Y^r] &= \int_{-\infty}^{\infty} x^r \mathrm{d}F_Y(x) \\ &= \int_{-\infty}^{\infty} x^r \mathrm{d}(F_{X_1}F_{X_2}) \\ &= \int_{-\infty}^{\infty} x^r F_{X_1}(x) f_{X_2}(x)\mathrm{d}x + \int_{-\infty}^{\infty} x^r F_{X_2}(x) f_{X_1}(x)\mathrm{d}x. \end{aligned} \tag{5.6}$$

Again, in general, the right-hand side integration is fundamentally impossible to solve analytically in such a way that $\mathsf{E}[Y^r]$ can be expressed directly as function of $\mathsf{E}[X_i^r]$.

In this chapter, we introduce the use of generalized lambda distributions (GLD) to allow the application of our moment method to parallel composition. The choice is based on three reasons. First and most importantly, due to the specific formulation of the GLD we can solve the integration problem in Eq. (5.4) so that it becomes straightforward to obtain the moments of Y. Second, unlike other approximating distributions such as the Pearson and Johnson distributions, the GLD uses only one function (see Eq. (5.7)) and is computationally simpler [83]. Third, the GLD covers a wide range of parameter values which are frequently used in modeling workloads, e.g., well-known and symmetric distributions.

Based on the use of the GLD as intermediate representation, the entire analysis approach for N-ary and binary parallel compositions is described in Figure 5.1. Given the first four input moments $\mathsf{E}[X_i^r]$, the GLD parameters (lambda values) can be evaluated by table look-up [83] or running a Nelder-Mead (NM) simplex procedure [72] for more precision. By using the lambda (and N) values in our analysis method, we obtain the resulting moments $\mathsf{E}[Y^r]$ for N-ary and binary parallel compositions, respectively.

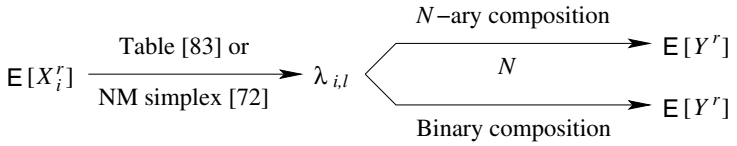

Figure 5.1: Performance analysis of parallel composition

This chapter is organized as follows. In the next section, we describe the generalized lambda distribution (GLD) as intermediate representation for our moment approach. Using GLD in Sections 5.2 and 5.3, we derive $\mathsf{E}[Y^r]$ for N-ary and binary parallel composition, respectively. In Sections 5.5 and 5.6, we evaluate the accuracy of our moment approach using synthetic as well as real programs, respectively.

5.1 Generalized Lambda Distribution

The generalized lambda distribution (GLD) is a four-parameter distribution defined by the percentile function R as function of the cdf, $0 \leq F \leq 1$, according to [83]

$$X = R_X(F) = \lambda_1 + \frac{F^{\lambda_3} - (1-F)^{\lambda_4}}{\lambda_2}, \tag{5.7}$$

where λ_1 is a location parameter, λ_2 is a scale parameter and λ_3 and λ_4 are shape parameters. Eq. (5.7) is also known as the Ramberg-Schmeiser (RS) parameterization of GLD[1].

While the cdf does not exist in closed form, the pdf is given by

$$f(x) = \frac{\mathrm{d}F}{\mathrm{d}R_X(F)} = \frac{1}{R'_X(F)} = \frac{\lambda_2}{\lambda_3 F^{\lambda_3 - 1} + \lambda_4 (1-F)^{\lambda_4 - 1}}. \tag{5.8}$$

This four-parameter distribution includes a wide range of curve shapes. Specifically, the distribution can provide good approximations to well-known densities as shown in Table 5.1 [83]. In the table, the GLD can exactly represent the continuous uniform

λ_r	U(a,b)	E(θ)	N(μ,σ)
λ_1	$(a+b)/2$	$(1+7.100e{-}3)/\theta$	μ
λ_2	$2/(b-a)$	$(-1.081e{-}3)\theta$	$0.1975/\sigma$
λ_3	1	$-4.070e{-}6$	0.1349
λ_4	1	$-1.076e{-}3$	0.1349

Table 5.1: λ values for well-known distributions.

distribution with sample space $[a,b]$, denoted U(a,b) (the second column of Table 5.1). The limit form of the GLD can represent the exponential distribution with parameter θ (i.e., $\mathsf{E}[X] = 1/\theta$), denoted E(θ), as $\lambda_4 \to 0$ when $\lambda_1 = \lambda_3 = 0$ and $\lambda_2 = \lambda_4/\theta$. While those lambda values provide a good approximation to E(θ) [83], for our experiments the lambda values in the third column of Table 5.1 are of more practical use. Another example is an approximation to the normal distribution, denoted N(μ,σ), where the lambda values are given in the fourth column of Table 5.1. The maximum error of the standard cdf $\Phi(x)$ is $\max_x \mid \Phi(x) - R^{-1}(x) \mid \approx 10^{-3}$ as found in [83]. Hence, the GLD can be used to approximate a wide range of distributions of which the first-four statistical moments are known.

Provided that the lambda values are given, obtaining the statistical moments proceeds as follows. Without loss of generality let $\lambda_1 = 0$. Then from Eqs. (3.9) and (5.7) the rth raw moment of X is given by

$$\mathsf{E}[X^r] = \frac{1}{\lambda_2^r} \sum_{i=0}^{r} \binom{r}{i} (-1)^i \mathrm{B}(\lambda_3(r-i)+1, \lambda_4 i + 1), \tag{5.9}$$

where B denotes the beta function as defined by

$$\mathrm{B}(a,b) = \int_0^1 t^{a-1}(1-t)^{b-1}\mathrm{d}t. \tag{5.10}$$

Since the beta function has been provided in current standard mathematical libraries, the computation cost of Eq. (5.9) for r up to four is negligible.

[1] Another GLD parametrization is also available, e.g., the Freimer, Mudholkar, Kollia and Lin (FMKL) representation [26]. Due to incompatibility with our analysis approach we do not consider the FMKL representation.

There are many approaches to obtain the lambda values: moment matching [83], probability-weighted moment estimates [44], least square [73], starship [52], percentiles [49], and quasi-random Sobol sequences [55]. Since in our analysis the moment values are available, the moment matching method is the most appropriate to use. In [83] it is shown how to obtain the four lambda values from the moments using a table. If more precise lambda values are required, an alternative to looking up the table is to apply a method based on function minimization such as the Nelder-Mead simplex procedure [72]. This procedure requires a constant number of iteration steps. Even for a 10^{-10} precision for the lambda values, our experiments show that no more than 100 iteration steps are required in evaluating Eq. (5.9) (i.e., in total less than 10^5 floating point operations). Thus in providing the first four moments, the cost of obtaining the lambda values from the table is also negligible.

5.2 N-ary Parallel Composition

In this section we present how the GLD is applied in the analysis of N-ary parallel composition, as presented in terms of Theorem 5.1.

Theorem 5.1 The nth order statistic
Let $Y_1 \leq Y_2 \leq \ldots \leq Y_N$ be random variables obtained by permuting N iid variates of continuous random variables X, i.e., $X_1, X_2, \ldots, X_N$, in increasing order. Let $\mathsf{E}[X^r]$, $r = 1, 2, 3, 4$ exists while X can be expressed in terms of GLD$(\lambda_1, \lambda_2, \lambda_3, \lambda_4)$ where λ_j are functions of $\mathsf{E}[X^r]$. Without loss of generality let $\lambda_1 = 0$. Then for $n = 1, 2, \ldots, N$ the rth raw moment of Y_n is given by

$$\mathsf{E}[Y_n^r] = \frac{n}{\lambda_2^r}\binom{N}{n}\sum_{i=0}^{r}\binom{r}{i}(-1)^i \mathrm{B}(\lambda_3(r-i)+n, \lambda_4 i + N - n + 1), \tag{5.11}$$

where B denotes the beta function.

Proof:
The rth raw moment of Y_n, expressed in terms of its cdf, is given by the Stieltjes integral [104]

$$\mathsf{E}[Y_n^r] = \int_{-\infty}^{\infty} x^r \mathrm{d}F_{Y_n}(x), \tag{5.12}$$

where $F_{Y_n}(x)$, expressed in terms of $F_X(x)$, is given by [104]

$$F_{Y_n}(x) = \sum_{k=n}^{N}\binom{N}{k}(F_X(x))^k(1-F_X(x))^{N-k}. \tag{5.13}$$

The derivative of Eq. (5.13) with respect to x is given by

$$\frac{\mathrm{d}F_{Y_n}(x)}{\mathrm{d}x} = \sum_{k=n}^{N}\binom{N}{k}\left(k(F_X(x))^{k-1}(1-F_X(x))^{N-k}\frac{\mathrm{d}F_X(x)}{\mathrm{d}x} - (N-k)(F_X(x))^k(1-F_X(x))^{N-k-1}\frac{\mathrm{d}F_X(x)}{\mathrm{d}x}\right)$$

$$
\begin{aligned}
&= N \sum_{k=n}^{N} \binom{N-1}{k-1} (F_X(x))^{k-1} (1 - F_X(x))^{N-k} \frac{\mathrm{d}F_X(x)}{\mathrm{d}x} \\
&\quad - N \sum_{k=n+1}^{N} \binom{N-1}{k-1} (F_X(x))^{k-1} (1 - F_X(x))^{N-k} \frac{\mathrm{d}F_X(x)}{\mathrm{d}x} \\
&= N \binom{N-1}{n-1} (F_X(x))^{n-1} (1 - F_X(x))^{N-n} \frac{\mathrm{d}F(x)}{\mathrm{d}x}. \qquad (5.14)
\end{aligned}
$$

Since the GLD is conveniently expressed as function of the cdf, i.e., $X = R(F)$, we substitute all variables in the right-hand side by the cdf. As $F_X = F$, x^r may be directly substituted by $(R(F))^r$. Consequently, the lower and upper bound follow $0 \leq F \leq 1$ since $F_{X_i}(x) = 0$ for $x \to -\infty$ and $F_{X_i}(x) = 1$ for $x \to \infty$. Substituting Eq. (5.14) and the lower and upper bounds to Eq. (5.12), it follows

$$
\mathsf{E}[Y_n^r] = N \binom{N-1}{n-1} \int_0^1 (R(F))^r F^{n-1} (1 - F)^{N-n} \mathrm{d}F, \qquad (5.15)
$$

where $(R(F))^r$ can be expanded into a power series according to

$$
(R(F))^r = \frac{1}{\lambda_2^r} \sum_{i=0}^{r} \binom{r}{i} (-1)^i F^{\lambda_3(r-i)} (1 - F)^{\lambda_4 i}. \qquad (5.16)
$$

Substituting Eq. (5.16) within Eq. (5.15) yields

$$
\begin{aligned}
\mathsf{E}[Y_n^r] &= \frac{N}{\lambda_2^r} \binom{N-1}{n-1} \int_0^1 \sum_{i=0}^{r} \binom{r}{i} (-1)^i F^{\lambda_3(r-i)+n-1} (1 - F)^{\lambda_4 i + N - n} \mathrm{d}F \\
&= \frac{N}{\lambda_2^r} \binom{N-1}{n-1} \sum_{i=0}^{r} \binom{r}{i} (-1)^i \int_0^1 F^{\lambda_3(r-i)+n-1} (1 - F)^{\lambda_4 i + N - n} \mathrm{d}F.
\end{aligned}
$$

Reducing the integral to a beta function results in

$$
\mathsf{E}[Y_n^r] = \frac{N}{\lambda_2^r} \binom{N-1}{n-1} \sum_{i=0}^{r} \binom{r}{i} (-1)^i \mathrm{B}(\lambda_3(r-i) + n, \lambda_4 i + N - n + 1).
$$

□

Theorem 5.1 enables us to express $\mathsf{E}[Y_n^r]$ in terms of $\mathsf{E}[X^r]$ and the problem size N (the number of tasks involved in the parallel section) while the value of n ranges from 1 to N.

5.2.1 N-ary And-Parallel Composition

As in and-parallel composition Y is determined by the largest X_i, the specific instance $n = N$ of Theorem 5.1 applies (i.e., the largest order statistic). Our result is stated in the following corollary.

Corollary 5.1 N-ary parallel composition
Under the same assumption as in Theorem 5.1 let random variable Y be defined as

$$
Y = \max(X_1, X_2, \ldots, X_N). \qquad (5.17)
$$

Then the rth raw moment of Y is given by

$$\mathsf{E}[Y^r] = \frac{N}{\lambda_2^r} \sum_{i=0}^{r} \binom{r}{i} (-1)^i \mathrm{B}(\lambda_3(r-i) + N, \lambda_4 i + 1). \tag{5.18}$$

□

The proof is obtained from Eq. (5.11) by substituting $n = N$. Similar to sequential compositions the solution complexity of Eq. (5.18) is entirely *independent* from N (i.e., $O(1)$) while its computation cost is negligible (cf. Eq. (5.9)).

5.2.2 N-ary Or-Parallel Composition

In this section we present how the GLD is applied in the analysis of speculative, N-ary or-parallel composition.

As in speculative parallel composition Y is determined by the smallest X_i, the specific instance $n = 1$ of Theorem 5.1 applies (i.e., the smallest order statistic). Our result is stated in the following corollary.

Corollary 5.2 N-ary Or-parallel composition
Under the same assumption as in Theorem 5.1 let random variable Y be defined as

$$Y = \min(X_1, X_2, \ldots, X_N). \tag{5.19}$$

Then the rth raw moment of Y is given by

$$\mathsf{E}[Y^r] = \frac{N}{\lambda_2^r} \sum_{i=0}^{r} \binom{r}{i} (-1)^i \mathrm{B}(\lambda_3(r-i) + 1, \lambda_4 i + N). \tag{5.20}$$

The proof can be straightforward obtained from Eq. (5.11) by substituting $n = 1$. Again, similar to sequential compositions the solution complexity of Eq. (5.20) is entirely *independent* from N (i.e., $O(1)$) while its computation cost is negligible (cf. Eq. (5.9)).

5.3 Binary Parallel Composition

While in the previous section, we consider X_i to be identical (thus narrowing the application), in this section, we present the analysis of binary parallel composition where X_i may be different. In the following, we discuss and-parallel and or-parallel binary composition separately.

5.3.1 Binary And-Parallel Composition

Recall the problem related to binary parallel composition in Eq. (5.6). As mentioned in Chapter 2, a number of heuristic approaches have been proposed as a result of the difficulties in finding a low-cost, accurate, analytical solution to solving the binary parallel composition problem. A good example of such an approach is found in [94]. Y is calculated by simply choosing X_i with the largest mean or by selecting the stochastic value with the

largest magnitude value in its entire range. For example (adapted from [94]), consider the maximum of $X_1 = 4 \pm 0.5$ and $X_2 = 3 \pm 2$. X_1 has the largest mean, and X_2 has the largest value within its range. On average however, the values of X_1 are likely to be higher than the values of X_2. We formulate the heuristic as described in [94] as follows

$$Y = \begin{cases} X_1, & \mathsf{E}[X_1] > \mathsf{E}[X_2] \\ X_1, & \mathsf{E}[X_1] = \mathsf{E}[X_2] \text{ and } \mathsf{E}[X_1^2] > \mathsf{E}[X_2^2] \\ X_2, & \text{otherwise.} \end{cases} \tag{5.21}$$

Being a simple heuristic, despite its attractive low-cost property, Eq. (5.21) only takes into account the first two moments $\mathsf{E}[X_i^r]$, and, amongst other things, does not compute the offset in $\mathsf{E}[Y]$ as established by the order statistics. Nevertheless, next to our positive experience with generalized lambda distributions, this heuristic has partly been the inspiration for our low-cost, analytic solution to the binary parallel composition problem.

While Theorem 5.1 and associated corollary restrict the workload X be iid, for *different* workload in binary and-parallel composition we present the following theorem.

Theorem 5.2 Binary And-parallel composition
Let random variable Y be defined as

$$Y = \max(X_1, X_2)$$

where X_i are independent random variables for which $\mathsf{E}[X_i^r]$, $r = 1, 2, 3, 4$ exists. Let X_i be expressed in terms of the GLD($\lambda_{i,1}$,$\lambda_{i,2}$,$\lambda_{i,3}$,$\lambda_{i,4}$) where $\lambda_{i,j}$ are functions of $\mathsf{E}[X_1^r]$ and $\mathsf{E}[X_2^r]$. Then the rth raw moment of Y is given by

$$\mathsf{E}[Y^r] = \int_0^1 (R_{X_2}(F))^r F_{X_1}(R_{X_2}(F))\mathrm{d}F + \int_0^1 (R_{X_1}(F))^r F_{X_2}(R_{X_1}(F))\mathrm{d}F. \tag{5.22}$$

Proof:
The rth raw moment of Y, expressed in terms of its cdf, is given by the Stieltjes integral [104]

$$\mathsf{E}[Y^r] = \int_{-\infty}^{\infty} x^r \mathrm{d}F_Y(x), \tag{5.23}$$

where $F_Y(x)$, expressed in terms of $F_{X_i}(x)$, is given by [104]

$$F_Y(x) = F_{X_1}(x) F_{X_2}(x). \tag{5.24}$$

The derivative of Eq. (5.24) with respect to x is given by

$$\frac{\mathrm{d}F_Y(x)}{\mathrm{d}x} = F_{X_1}(x)\frac{\mathrm{d}F_{X_2}(x)}{\mathrm{d}x} + F_{X_2}(x)\frac{\mathrm{d}F_{X_1}(x)}{\mathrm{d}x}. \tag{5.25}$$

Since the GLD is conveniently expressed as function of the cdf, we substitute all variables in the right-hand side by the cdf. As $F_{X_i} = F$, x may be directly substituted by $R_{X_i}(F)$. Consequently, the lower and upper bound follow $0 \leq F \leq 1$ since $F_{X_i}(x) = 0$ for $x \to -\infty$ and $F_{X_i}(x) = 1$ for $x \to \infty$. Substituting Eq. (5.25) and the lower and upper bounds to Eq. (5.23), it follows

$$\mathsf{E}[Y^r] = \int_0^1 (R_{X_2}(F))^r F_{X_1}(R_{X_2}(F))\mathrm{d}F + \int_0^1 (R_{X_1}(F))^r F_{X_2}(R_{X_1}(F))\mathrm{d}F.$$

□

Note that Eq. (5.22) indeed expresses the commutative property of the binary parallel composition.

Eq. (5.22) shows that $\mathsf{E}[Y^r]$ can be evaluated from the distributions of X_1 and X_2 by efficiently computing the integral from $F = 0$ to $F = 1$ rather than from $x = -\infty$ to $x = \infty$. Unlike Eq. (5.18) for N-ary parallel composition, in Eq. (5.22), $F_{X_i}(R_{X_j}(F))$ poses a computation problem since the cdf of GLD cannot be expressed explicitly in terms of λ. Hence, in the following, we describe three approximation techniques to compute Eq. (5.22). The first approach is based on a graphical interpretation. The second is based on the generalized beta distribution. Finally, we introduce a solution based on Newton's method. The three methods represent different trade-offs between cost and accuracy.

5.3.2 Graphical Interpretation Method

To illustrate the principle of our approach, we consider the following example. Let X_1 and X_2 be uniformly distributed random variables given by

$$F_{X_1} = \begin{cases} 0, & x < 0 \\ mx, & 0 \leq x < 1/m \\ 1, & x > 1/m, \end{cases} \tag{5.26}$$

where m is a shape parameter, and

$$F_{X_2} = \begin{cases} 0, & x < 1 \\ 2(x-1), & 1 \leq x < 3/2 \\ 1, & x > 3/2. \end{cases} \tag{5.27}$$

Figure 5.2 shows a trivial case for $m = 1$. From Eq. (5.5) we immediately obtain $F_Y = F_{X_2}$ since $F_{X_1} = 1$ for $F_{X_2} > 0$. In contrast, Figure 5.2 (right) shows the resulting F_Y (dashed line) using Eq. (5.5) for $m = 1/2$. Due to the integration problem of Eq. (5.6) this exact solution F_Y cannot be evaluated explicitly in terms of $\mathsf{E}[X_i^r]$. Applying heuristic Eq. (5.21) would simply yield $Y = X_2$ independent of $\mathsf{Var}[X_1]$. Consequently this heuristic causes a large estimation error shown by area between F_{X_2} and F_Y which increases monotonically as function of $\mathsf{Var}[X_1]$.

A better approximation can be obtained by taking the *minimum* of F_{X_1} and F_{X_2}:

$$F_Y = \min(F_{X_1}, F_{X_2}), \tag{5.28}$$

as shown by the bold solid line in Figure 5.2 (right). In contrast to the heuristic method in Eq. (5.21), this approach implicitly takes $\mathsf{Var}[X_1]$ into account such that the estimation error is less sensitive to $\mathsf{Var}[X_1]$. Using Eq. (5.28), we can compute $\mathsf{E}[Y^r]$ in Eq. (5.22). Let F_0 be the value of F at the intersection of F_{X_1} and F_{X_2} (see Figure 5.2 (right)). A straightforward implementation is given by evaluating the integration for $0 \leq F \leq F_0$ and $F_0 \leq F \leq 1$ according to

$$\mathsf{E}[Y^r] = \int_0^{F_0} (R_{X_2}(F))^r F_{X_1}(R_{X_2}(F))\ \mathrm{d}F + \int_{F_0}^1 (R_{X_1}(F))^r F_{X_2}(R_{X_1}(F))\ \mathrm{d}F. \tag{5.29}$$

Eq. (5.29) is straightforward to compute since each integration involves one cdf (cf. Eq. (5.22). However, Eq. (5.29) still suffers from inaccuracy for $\mathsf{E}[X_1] = \mathsf{E}[X_2]$.

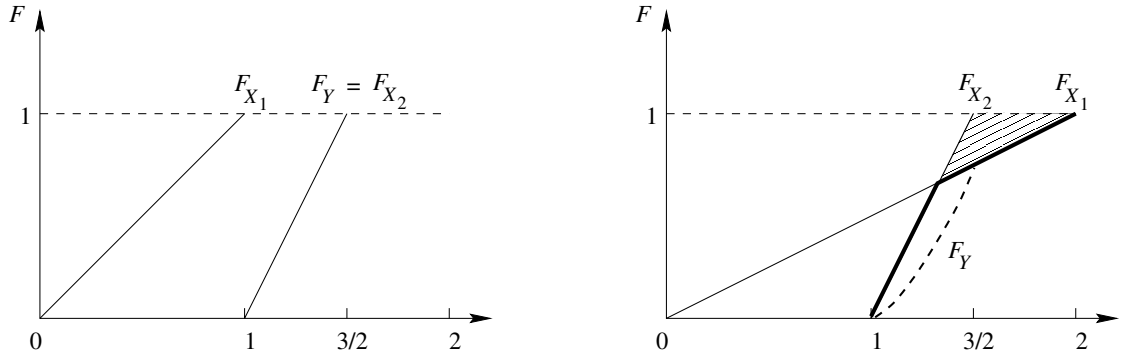

Figure 5.2: $Y = \max(X_1, X_2)$ for $m = 1$ (left) and $Y = \max(X_1, X_2)$ for $m = 1/2$ (right)

5.3.3 GBD Method

As mentioned before, the cdf of the GLD does not exist in closed-form, while in the analysis of binary parallel composition, a closed-form cdf is required. As an approximation, we can compute the cdf of X based on generalized beta distributions. The Generalized Beta Distributions (GBD) is a four-parameter distribution defined by the cdf

$$F_X(x) = \int_0^{\frac{x-\beta_1}{\beta_2}} \frac{t^{\beta_3-1}(1-t)^{\beta_4-1}}{\mathrm{B}(\beta_3,\beta_4)}\mathrm{d}t = I_{(x-\beta_1)/\beta_2}(\beta_3,\beta_4), \tag{5.30}$$

where β_1 is a location parameter, β_2 is a scale parameter and $\beta_3 > 0$ and $\beta_4 > 0$ are shape parameters, and $I_x(a,b)$ is known as incomplete beta function. Similar to the GLD, the distribution can provide good approximations to well-known distributions [50].

All the raw moments of X exist and can be derived from the relation $X = \beta_1 + \beta_2\bar{X}$ where $\bar{X}$ denotes a beta distribution whose the rth moment is given by the following convenient recursive expression

$$\mathsf{E}[\bar{X}^r] = \frac{\beta_3 + r - 1}{\beta_3 + \beta_4 + r - 1}\mathsf{E}[\bar{X}^{r-1}]. \tag{5.31}$$

From Eq. (5.31) the raw moments of GBD are given by

$$\mathsf{E}[X^r] = \mathsf{E}[(\beta_1 + \beta_2\bar{X})^r] = \sum_{k=0}^{r}\binom{r}{k}\beta_1^k\beta_2^{r-k}\mathsf{E}[\bar{X}^{r-k}]. \tag{5.32}$$

GBD covers a great diversity of shapes and has found wide application in modeling measurements that are restricted to a finite interval. Although we can refer to a table as found in [50], unlike the GLD, obtaining a solution for the β values given the first four moments can be quite difficult [50]. Yet, in providing the first four moments, the cost of obtaining the β values are typically small. Table 5.2 shows β for $\mathrm{U}(a,b)$, $\mathrm{E}(\theta)$, and $\mathrm{N}(\mu,\sigma)$. While the β values are exact for $\mathrm{U}(a,b)$, for $\mathrm{E}(\theta)$, and $\mathrm{N}(\mu,\sigma)$, the β's are valid for values of L that are sufficient large from the mean. Our results show that $L = 10^3$ is sufficiently large.

Table 5.2: β values for well-known distributions for $L \to \infty$

β	U(a,b)	E(θ)	N(μ, σ)
β_1	a	0	$\mu - \sigma\sqrt{2L}$
β_2	$b-a$	L/θ	$\sigma\sqrt{8L}$
β_3	1	1	L
β_4	1	L	L

To compute $\mathsf{E}[Y^r]$ in Eq. (5.22), we use Eq. (5.30) for evaluating $F_{X_i}(R_{X_j}(F))$ where $R_{X_j}(F)$ is evaluated using GLD. An implementation of Eq. (5.22) is given by

$$\mathsf{E}[Y^r] = \frac{1}{K}\sum_{k=1}^{K}((R_{X_2}(\frac{k}{K}))^r F_{X_1}(R_{X_2}(\frac{k}{K})) + (R_{X_1}(\frac{k}{K}))^r F_{X_2}(R_{X_1}(\frac{k}{K}))). \tag{5.33}$$

Although Eq. (5.33) yields accurate values for $\mathsf{E}[Y^r]$, obtaining the β values can be quite difficult as mentioned before.

5.3.4 Newton's Method

Another interesting method to approximate the cdf of X is Newton's method, which is given by

$$F_{i+1} = F_i - \frac{R_X(F_i)}{R'_X(F_i)} \tag{5.34}$$

where i is the iteration index. The use of Newton's method is appropriate since the method converges quadratically and converges to a solution within a few iterations due to the fact that F_i is close to F_{i+1}. For instance, to compute a cdf for 10^4 intervals it requires 30 ms on a 1 GHz Pentium III processor.

As in Eq. (5.22) the integration proceeds from $F = 0$ to $F = 1$ rather than $x = -\infty$ to $x = \infty$, a straightforward implementation of Eq. (5.22) is given by the summation

$$\mathsf{E}[Y^r] = \frac{1}{K}\sum_{k=1}^{K}\left((R_{X_2}(k/K))^r F_{X_1}(R_{X_2}(k/K)) + (R_{X_1}(k/K))^r F_{X_2}(R_{X_1}(k/K))\right). \tag{5.35}$$

In practice, $K = 10^4$ is sufficiently large while $F_{X_i}(R_{X_j}(k/K))$ can be computed using Newton's method as the following.

$$F_{X_{i,l+1}}(R_{X_j}(k/K)) = F_{X_{j,l}} - R_{X_j}(F_{X_{j,l}})/R'_{X_j}(F_{X_{j,l}}) \tag{5.36}$$

where $F_{X_{j,0}} = 1/K$. The computation of Eq. (5.35) costs 60 ms on a 1 GHz Pentium III processor.

One might wonder why Eq. (5.22) would not suffice as the N-ary parallel composition might be implemented by the binary parallel composition. The reasons are that the related computation requires $\lceil {}^2\log(N) \rceil$ times evaluation of Eq. (5.22) and the accuracy of Eq. (5.22) is worst for iid workload.

5.4 Binary Or-Parallel Composition

For different workloads in binary *or*-parallel composition we present the following theorem.

Theorem 5.3 Binary Or-parallel composition
Let random variable Y be defined as

$$Y = \min(X_1, X_2) \tag{5.37}$$

where X_i are independent random variables for which $\mathsf{E}[X_i^r]$ for $r = 1, 2, 3, 4$ exist. Let X_i be expressed in terms of $\mathrm{GLD}(\lambda_{i,1}, \lambda_{i,2}, \lambda_{i,3}, \lambda_{i,4})$, where $\lambda_{i,j}$ are functions of $\mathsf{E}[X_i^r]$. Then the raw moments of Y are given by

$$\mathsf{E}[Y^r] = \int_0^1 (R_{X_1}(F))^r(1 - F_{X_2}(R_{X_1}(F))) + (R_{X_2}(F))^r(1 - F_{X_1}(R_{X_2}(F)))\mathrm{d}F. \tag{5.38}$$

Proof:
The rth raw moment of Y, expressed in terms of its cdf, is given by the Stieltjes integral [104]

$$\mathsf{E}[Y^r] = \int_{-\infty}^{\infty} x^r \mathrm{d}F_Y(x), \tag{5.39}$$

where $F_Y(x)$, expressed in terms of $F_{X_i}(x)$, is given by [104]

$$F_Y(x) = 1 - (1 - F_{X_1}(x))(1 - F_{X_2}(x)). \tag{5.40}$$

The derivative of Eq. (5.40) with respect to x is given by

$$\frac{\mathrm{d}F_Y(x)}{\mathrm{d}x} = (1 - F_{X_2}(x))\frac{\mathrm{d}F_{X_1}(x)}{\mathrm{d}x} + (1 - F_{X_1}(x))\frac{\mathrm{d}F_{X_2}(x)}{\mathrm{d}x}. \tag{5.41}$$

Since the GLD is conveniently expressed as function of the cdf, we substitute all variables in the right-hand side by the cdf. As $F_{X_i} = F$, x may be directly substituted by $R_{X_i}(F)$. Consequently the lower and upper bound follow $0 \leq F \leq 1$ since $F_{X_i}(x) = 0$ for $x \to -\infty$ and $F_{X_i}(x) = 1$ for $x \to \infty$. Expressed in terms of Eq. (5.39), it follows

$$\mathsf{E}[Y^r] = \int_0^1 (R_{X_1}(F))^r(1 - F_{X_2}(R_{X_1}(F))) + (R_{X_2}(F))^r(1 - F_{X_1}(R_{X_2}(F)))\mathrm{d}F.$$

□

A straightforward implementation of Eq. (5.38) is as follows

$$\mathsf{E}[Y^r] = \frac{1}{K}\sum_{k=1}^{K}((R_{X_1}(\frac{k}{K}))^r(1 - F_{X_2}(R_{X_1}(\frac{k}{K}))) + (R_{X_2}(\frac{k}{K}))^r(1 - F_{X_1}(R_{X_2}(\frac{k}{K})))) \tag{5.42}$$

where K is large. In practice, $K = 10^4$ is sufficiently large.

5.5 Synthetic Workloads

In this section we describe the quality of our prediction approach when applied to synthetic distributions and those measured from applications. The prediction quality is expressed in terms of the relative error ε_r which is defined according to

$$\varepsilon_r = \frac{|\mathsf{E}[Y_m^r] - \mathsf{E}[Y_p^r]|}{\mathsf{E}[Y_m^r]}, \tag{5.43}$$

where $\mathsf{E}[Y_m^r]$ and $\mathsf{E}[Y_p^r]$ are the measured and predicted moments, respectively. Synthetic distributions such as the uniform, exponential and normal distributions are frequently used for modeling workloads. The pdf of these distributions is available in close-form from which the moments $\mathsf{E}[Y_m^r]$ for parallel composition can be obtained. While for the uniform and exponential distributions $\mathsf{E}[Y_m^r]$ can be expressed explicitly in terms of the input moments, for normal distributions, however, $\mathsf{E}[Y_m^r]$ is evaluated from numerical integration since the cdf is not available in closed form. For synthetic distributions, $\mathsf{E}[Y_m^r]$ is obtained numerically using Maple Mathematical Software [65] since the pdf of Y_m is available in closed form. To determine $\mathsf{E}[Y_p^r]$ we use Eqs. (5.18) and (5.20) for N-ary and-parallel and or-parallel compositions, respectively, while for binary parallel composition based on graphical interpretation, GBD and Newton's methods we use Eqs. (5.29), (5.33) and (5.35), respectively. The input moments $\mathsf{E}[X_i^r]$ are also measured from 6,000 random samples. The prediction errors are presented in figures and finally summarized in tables.

5.5.1 Uniform Distribution

The first synthetic distribution is the continuous uniform distribution. When the input workloads X_i are continuous uniformly distributed, then $\mathsf{E}[Y_m^r]$ is exactly equal to $\mathsf{E}[Y_p^r]$ for N-ary parallel composition[2] since X_i are exactly determined by the GLD (see Table 5.1). For binary parallel composition, the error is also equal to zero. Such a good result is not achieved using the low-cost graphical interpretation method in Eq. (5.29) and the heuristic method in Eq. (5.21). Consider the following example. Let X_1 and X_2 be continuous uniform distributions defined by Eqs. (5.26) and (5.27), respectively. These distributions are also illustrated in Figure 5.2. Parameter m in X_1 is introduced to evaluate the relative error between X_1 and X_2 for various scenarios. For $m \geq 1$, both methods have no error since the cdf's are disjunct (i.e., $Y = X_2$). For $2/3 \leq m < 1$, both methods yield the same error while for $0.1 \leq m < 2/3$, the graphical interpretation method is much better than the heuristic method as shown in Figure 5.3. The maximum error of the graphical interpretation method is 15% while ε_1 is less than 5%.

5.5.2 Exponential Distribution

When X_i are exponentially distributed, our results for and-parallel composition are as follows. Figure 5.4 shows ε_r for $\theta = 1$ and N ranging from 2 to $1,000$ (numerical instabilities

[2]This applies to both and-parallel and or-parallel compositions due to symmetry of the pdf.

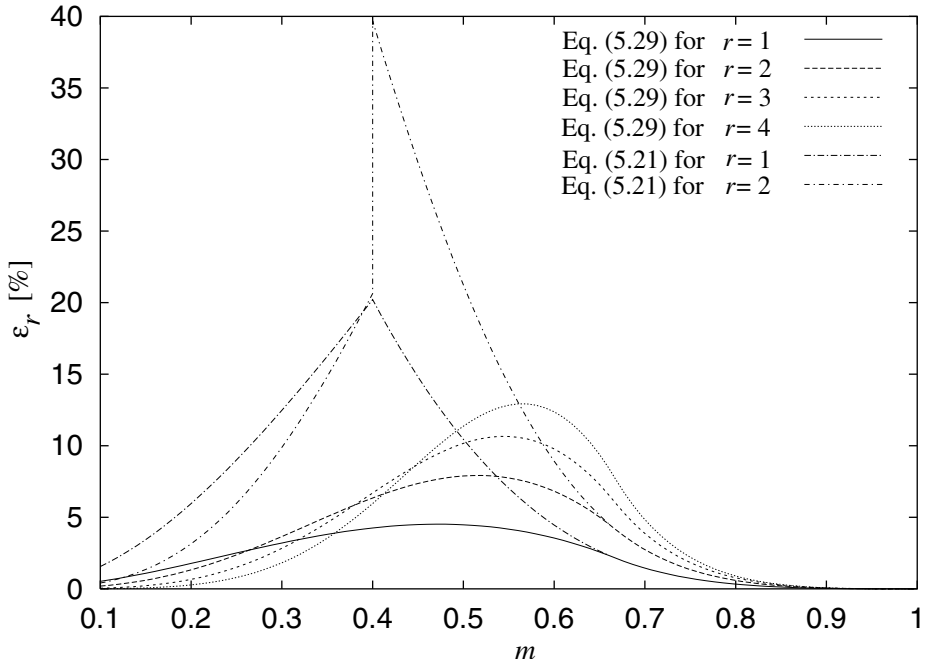

Figure 5.3: $\varepsilon_r[\%]$ for uniform distribution

in Maple prohibit larger values for N). The figure shows that the error decreases monotonically as a function of N because the GLD, in limit form, approaches the exponential distribution. The largest error, for $N = 2$, is less than 1%.

For binary parallel composition we consider two exponential distributions with parameters $\theta_1 = 1$ and $0.1 \leq \theta_2 \leq 10$. Figure 5.5 shows ε_r for binary and-parallel composition using graphical interpretation method. As a consequence of the prediction principle in Section 5.3.2, the error is the largest when $\mathsf{E}[X_1] = \mathsf{E}[X_2]$. In this case, the heuristic method in Eq. (5.21) has the same performance since $F_{X_2} < F_{X_1}$ in the entire range causes both methods to return X with the greatest mean value. Figures 5.6 and 5.7 show ε_r for binary and-parallel composition using GBD and Newton's methods, respectively. In contrast to Figure 5.5, these two figures show that ε_r is in the few percent range and insensitive to the θ_2 value variation. Specifically, ε_1 is less than 1%.

For or-parallel composition, we obtain similar results as shown by Figures 5.8 and 5.9. For binary or-parallel composition, we only present the results using Newton's method since the method outperform the others.

5.5.3 Normal Distribution

To study N-ary parallel composition using normal distributions we let $\mathsf{E}[X] = 0$ and $\mathsf{Var}[X] = 1$. Figure 5.10 shows ε_r for N up to $10,000$. Again, numerical instabilities prohibit larger values for N. In the figure we also compare our approach with Gumbel's approximation formula for general symmetric distributions as given in [36]

$$\mathsf{E}[Y_N] \approx \mathsf{E}[X] + \mathsf{Std}[X]\sqrt{2\log(0.4N)}. \tag{5.44}$$

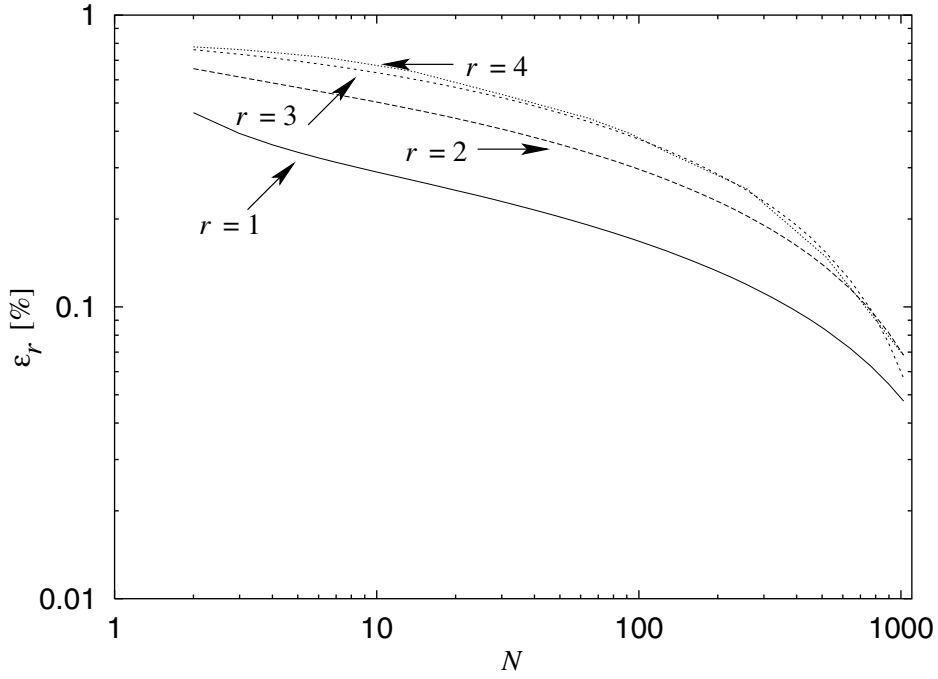

Figure 5.4: $\varepsilon_r[\%]$ for the exponential distribution (N-ary and-parallel)

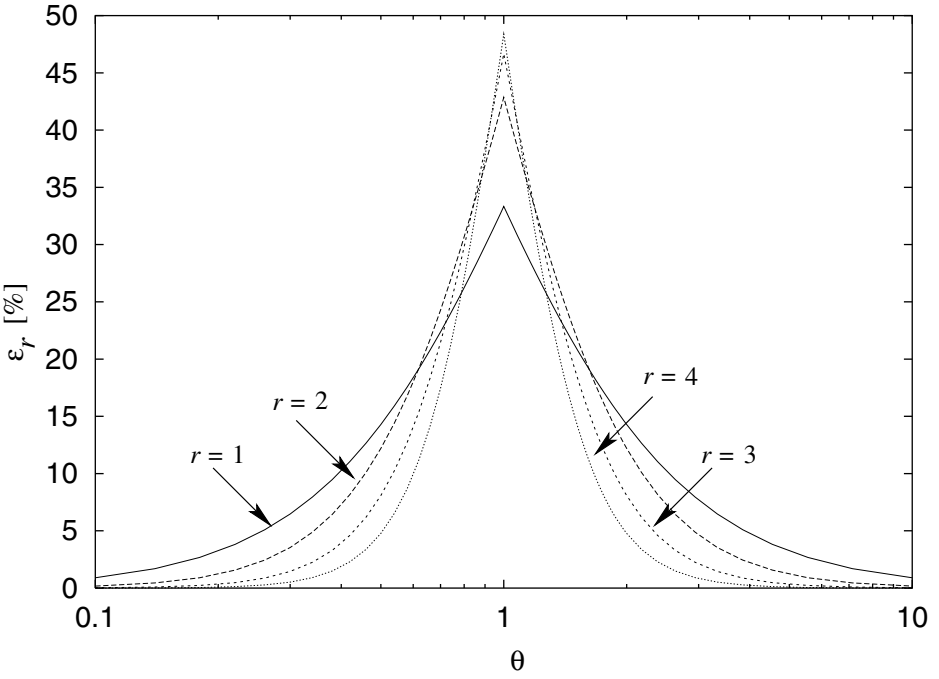

Figure 5.5: $\varepsilon_r[\%]$ for exponential distribution (binary and-parallel composition using graphical interpretation method

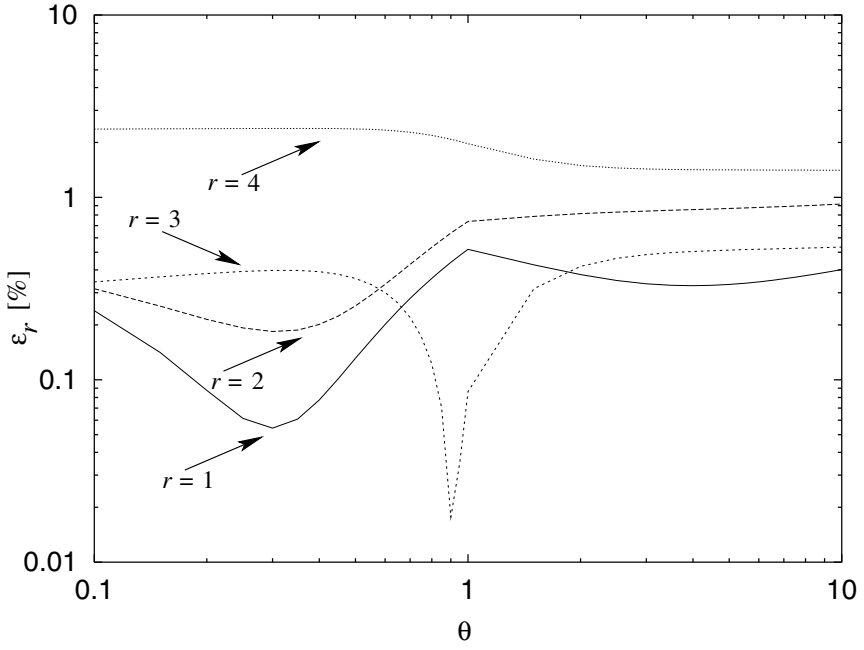

Figure 5.6: ε_r[%] for exponential distribution (binary and-parallel composition using GBD method)

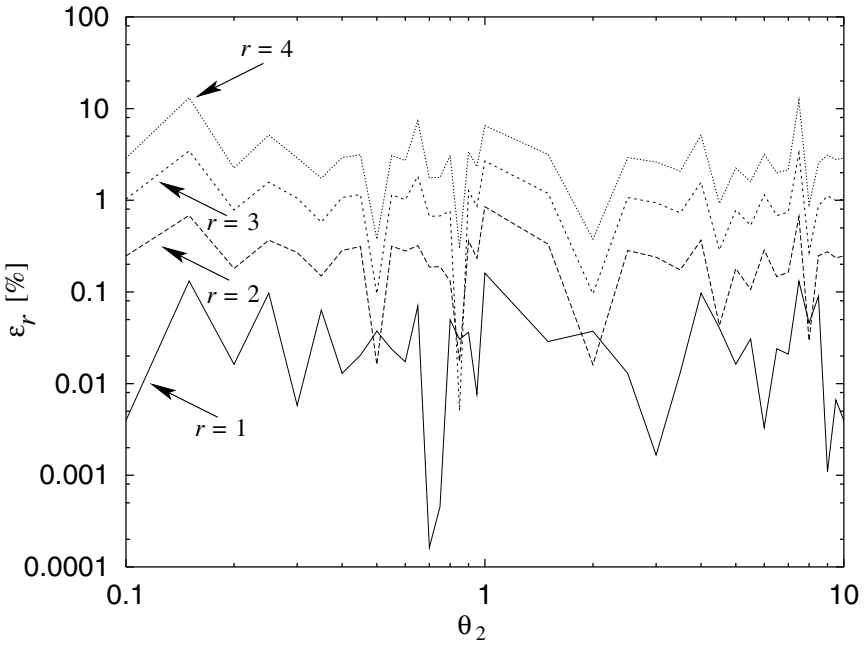

Figure 5.7: ε_r[%] for exponential distribution (binary and-parallel composition using Newton's method)

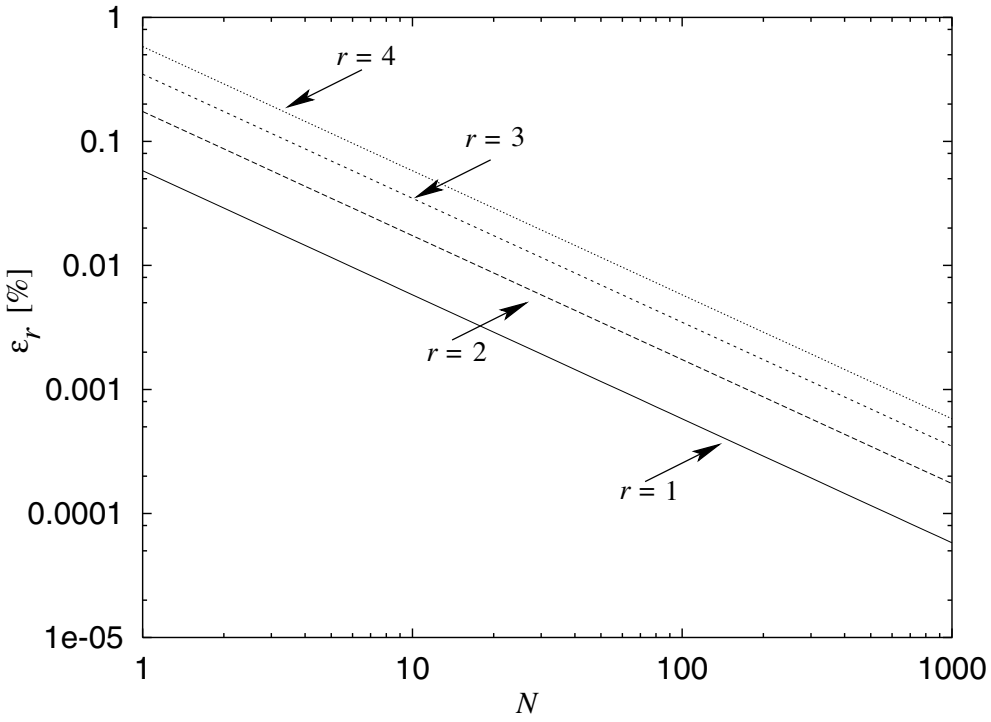

Figure 5.8: $\varepsilon_r[\%]$ for exponential distribution (N-ary or-parallel)

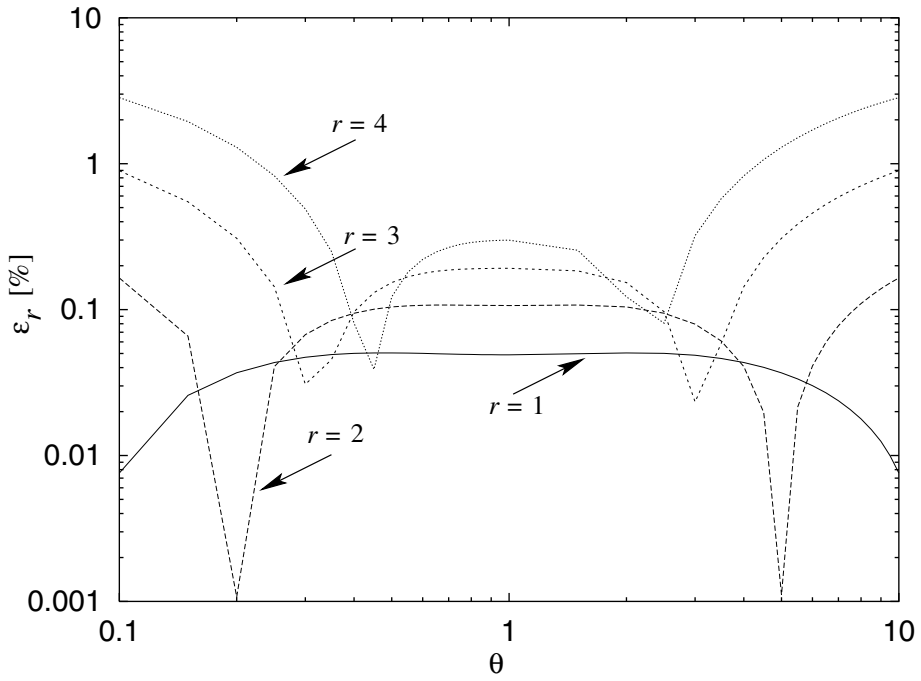

Figure 5.9: $\varepsilon_r[\%]$ for exponential distribution (binary or-parallel using Newton's method)

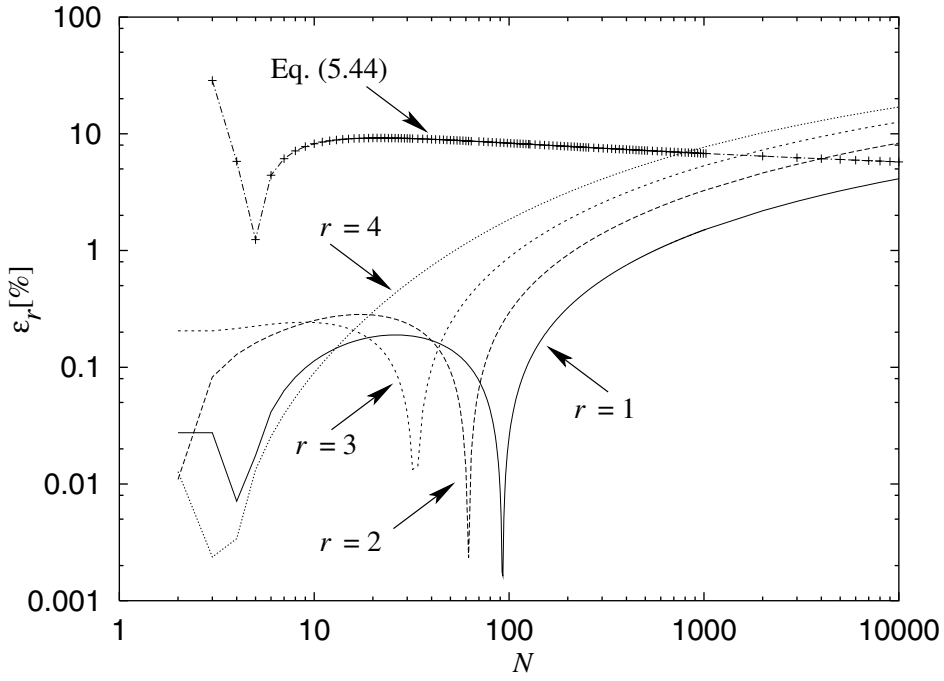

Figure 5.10: $\varepsilon_r[\%]$ for the normal distribution (N-ary and-parallel composition)

The error of Eq. (5.44) approaches zero very slowly for large N, while our approximation is much better for $N < 100$ and is still better for N up to $10,000$.

For binary and-parallel composition we consider two normal distributions where $\mathsf{E}[X_1] = \mathsf{Var}[X_1] = 1$, while $\mathsf{E}[X_2] = \mu$ and $\mathsf{Var}[X_2] = \sigma^2$. We vary μ and σ as shown in Figures 5.11 and 5.12, respectively, using the graphical interpretation method. Again, the prediction error decreases as the workloads diverge, while for larger variance, graphical interpretation method outperforms the heuristic in Eq. (5.21). In Figure 5.11, both methods have the same error since the predicted Y is X_2. In Figure 5.12, the decreasing ε_r for $r = 2$ and $\sigma = 2$ is due to $\mathsf{E}[Y^2] > \mathsf{E}[X_2^2]$ for $1 < \sigma < 2$ while $\mathsf{E}[Y^2] < \mathsf{E}[X_2^2]$ for $\sigma > 2$. The maximum error of $\mathsf{E}[Y]$ is 35%, while the error of graphical interpretation method decreases with $\mathsf{Var}[X_2]$ whereas the error of heuristic steadily increases.

A similar scenario is also used for the prediction error of binary and-parallel composition using GBD and Newton's methods as shown in Figures 5.13, 5.14, 5.15 and 5.16, respectively. In Figures 5.13 and 5.15, ε_r decreases logarithmically as function of μ while in Figures 5.14 and 5.16, ε_r increases logarithmically as function of σ. In all cases ε_r is below 1%. Numerical instabilities in the computation of $\mathsf{E}[Y_m^r]$ using Maple prohibit larger values for σ. These figures show that for the same mean, the error increases as the variance is large while for the same variance the largest error occurs when the mean is the same. The prediction error is much better than that using the graphical interpretation method.

For N-ary or-parallel composition, we obtain similar prediction error as shown in Figure 5.10 due to the fact that the normal distribution is symmetric. For binary or-parallel composition, the prediction error is shown in Figures 5.17 and 5.18 using Newton's

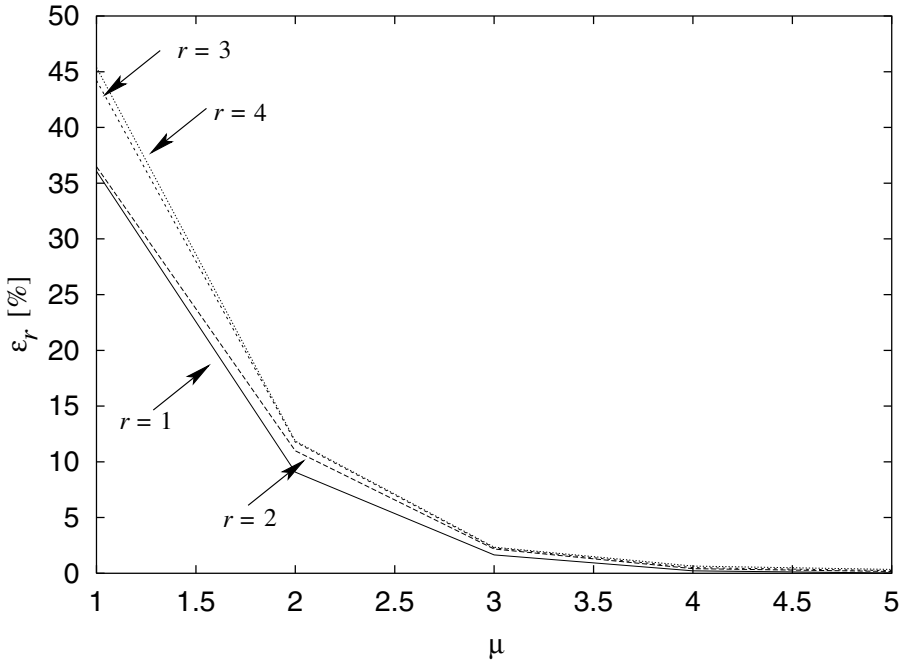

Figure 5.11: $\varepsilon_r[\%]$ for the normal distribution with $\sigma = 1$ (binary and-parallel composition using graphical interpretation method).

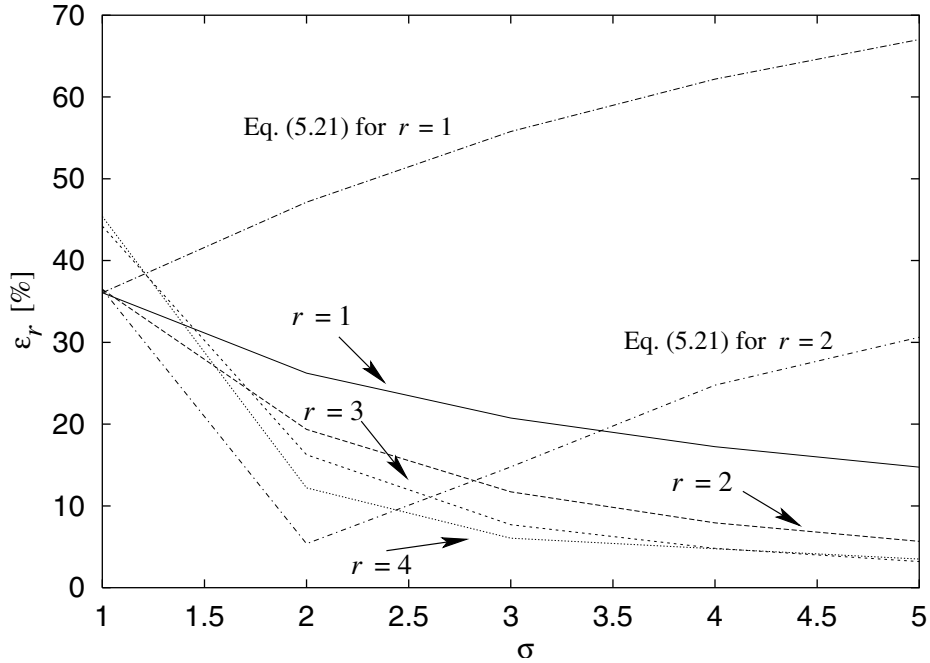

Figure 5.12: $\varepsilon_r[\%]$ for the normal distribution with $\mu = 1$ (binary and-parallel composition using graphical interpretation method).

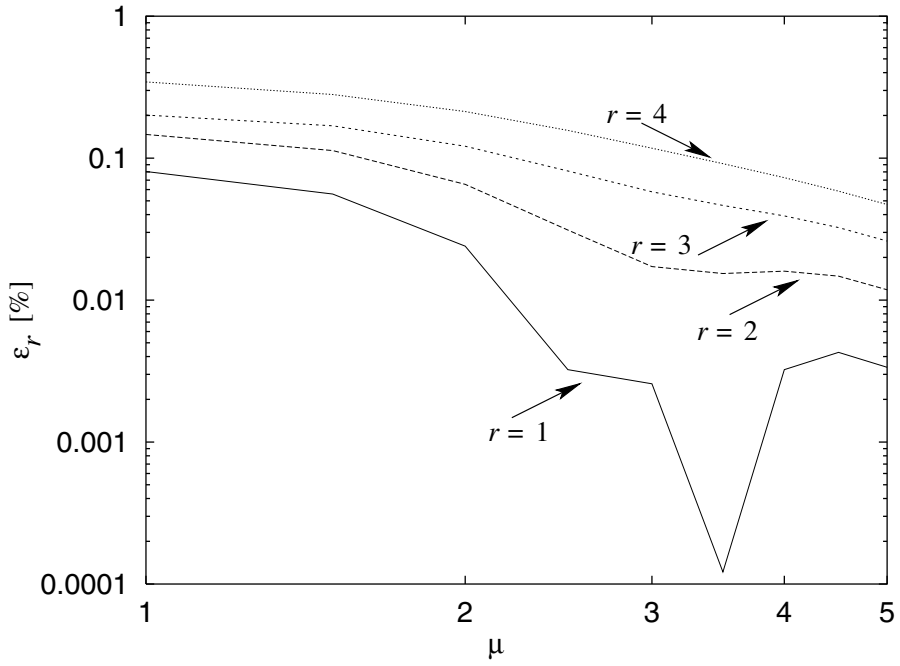

Figure 5.13: $\varepsilon_r[\%]$ for the normal distribution with $\sigma = 1$ (binary and-parallel composition using GBD method)

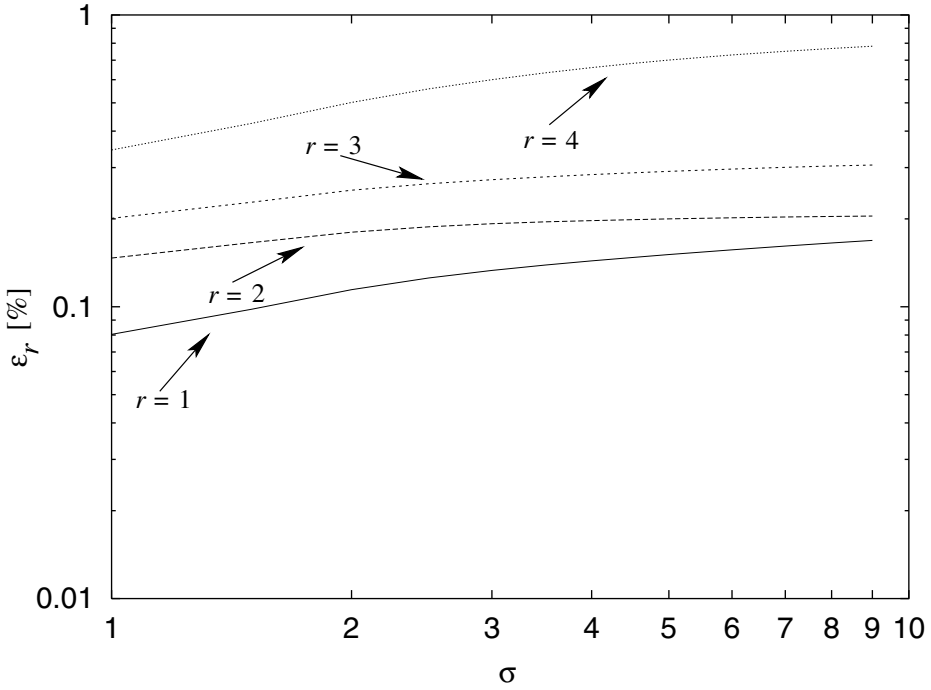

Figure 5.14: $\varepsilon_r[\%]$ for the normal distribution with $\mu = 1$ (binary and-parallel composition using GBD method)

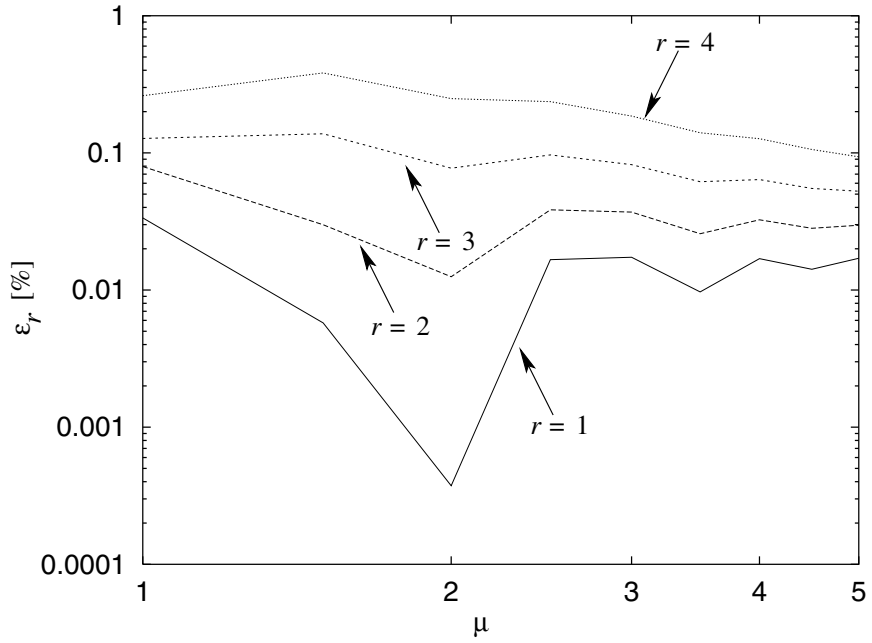

Figure 5.15: $\varepsilon_r[\%]$ for the normal distribution with $\sigma = 1$ (binary and-parallel composition using Newton's method)

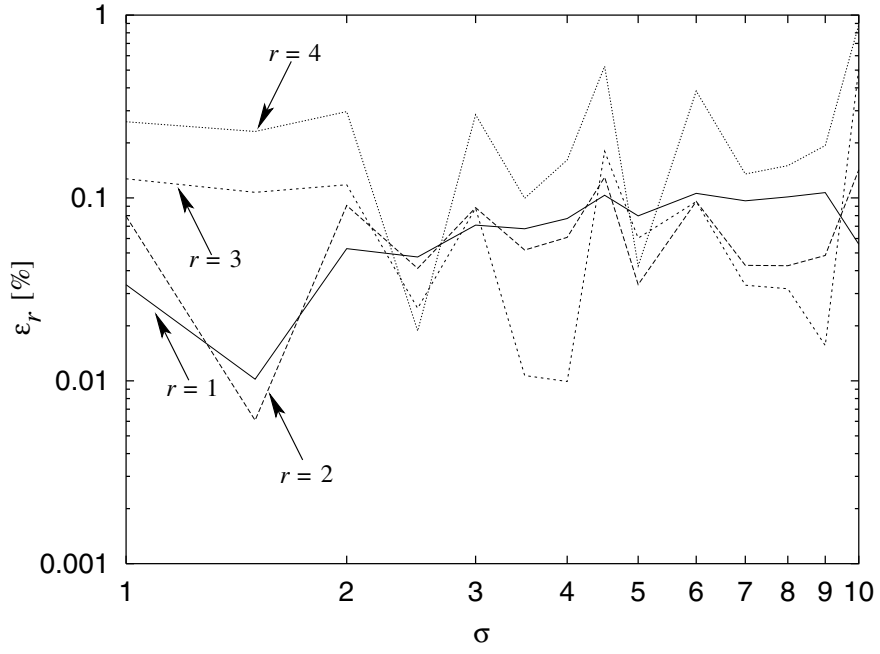

Figure 5.16: $\varepsilon_r[\%]$ for the normal distribution with $\mu = 1$ (binary and-parallel composition using Newton's method)

method. Again, we present Newton's method only since the prediction outperforms the other methods.

Summarizing this section, we conclude that the maximum error of our method is in the few percent range and insensitive to the variation in distributions. For binary parallel composition Newton's method outperforms the other two methods. Finally, we note that the reason for the error shape in Figures 5.3 to 5.18 is caused by numerical instabilities and the effects of the higher moments (5th, 6th, etc), that are not accounted for in our four-moments approach.

5.6 Empirical Workloads

In this section, we describe to what extent our approximation approach applied on workloads measured from real applications. Our study involves 6 applications. The first application, the NAS Embarrassingly Parallel benchmark (NAS-EP), is an example of identical and independent normal distributions occurring in practice. While standard distributions are often used to represent task execution time, in practice distributions may be totally different so that a representation based on standard distributions is inadequate. To illustrate this, we study a simple data parallel version of Single Source Shortest Path (SSSP) problem [82] where the measured workloads are lightly correlated. To investigate the effect of different levels of correlation between the tasks we also apply our analysis technique to a Parallel Sorting by Regular Sampling algorithm (PSRS) [97] which comprises three data parallel sections. Next to the correlation we also investigate to what extent our approach requires distributions to be unimodal. To this end, we apply our technique to WATOR, a data parallel simulation program in which extreme workload imbalances can occur. To investigate the accuracy using workload from task parallel programs we perform an experiment where two workloads are pipelined and form a binary and-parallel composition The above applications employ and-parallelism which is the predominant form of parallel composition. In order to evaluate or-parallelism, we study speculative parallel in the context of distributed web search-engine in which both binary and N-ary or-parallel scenarios are studied.

5.6.1 NAS-EP

The NAS benchmark suite has been developed for the performance evaluation of highly parallel supercomputers [8]. For our experiments we only consider the first parallel kernel, called the Embarrassingly Parallel kernel (EP), since the other benchmarks merely exhibit deterministic behavior due to the limited dependency of their workload on the input data. NAS-EP essentially generates M Gaussian deviates whose number is not constant on each run which implies a stochastic workload for each parallel task. This number is selected as our workload X_i.

In our experiment we consider an average workload for each processor corresponding to $M = 10^{20}$. Hence, the workload per processor can be assumed to be iid to which Eq. (5.18) applies. The benchmark is run $6,000$ times from which the same number of samples for $X_1, \ldots, X_N$ and Y is obtained. The moments of the workload are presented in Table 5.3, in terms of central moments for convenience. The results show that the input

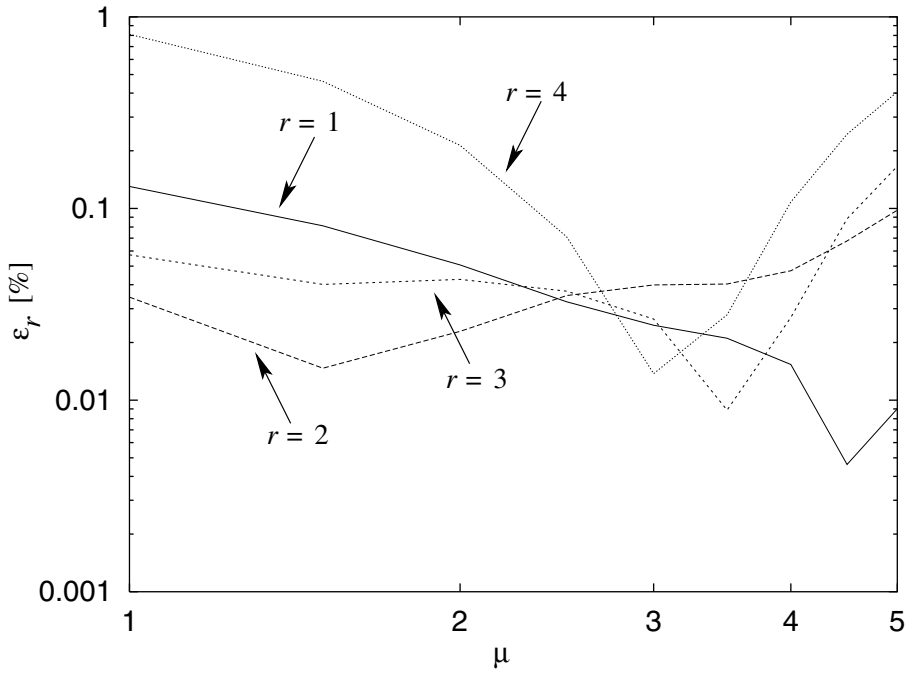

Figure 5.17: ε_r[%] for the normal distribution with $\sigma = 1$ (binary or-parallel composition using Newton's method)

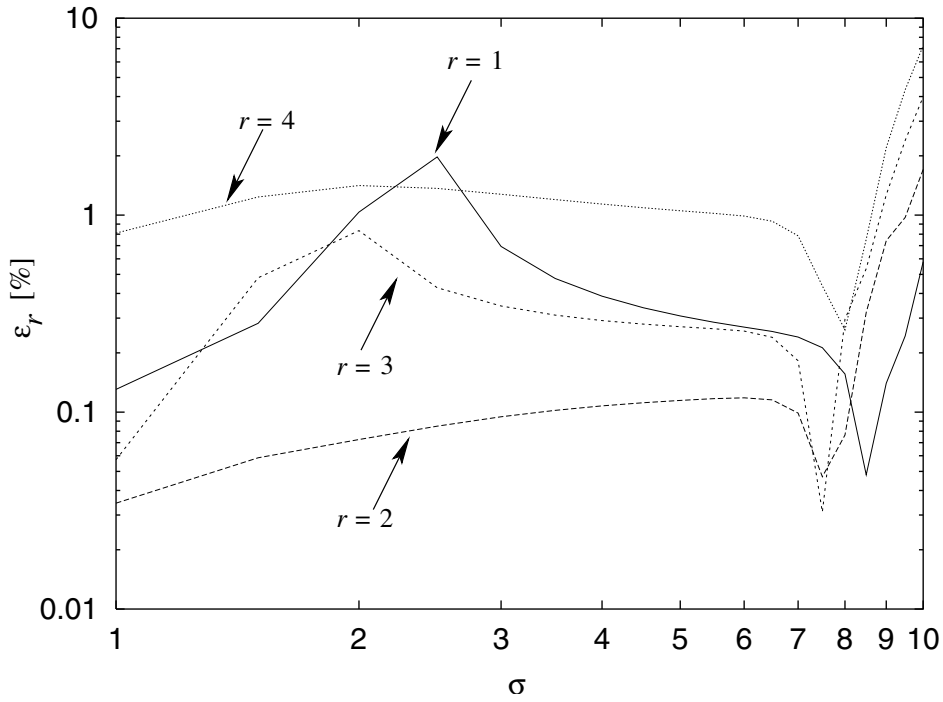

Figure 5.18: ε_r[%] for the normal distribution with $\mu = 1$ (binary or-parallel composition using Newton's method)

distribution (X_1) approximates the normal distribution based on the Chi-Square test (χ^2) (the χ^2 value of X_1 is 65.3 for a 5% significance level while the threshold is 67.5). The pdf of X_i is shown in Figure 5.19. Thus X_i is well-fitted to the normal distribution.

Table 5.3: The moments of X_1 for NAS-EP

E[X_1]	Var[X_1]	Skw[X_1]	Kur[X_1]
823,549	180,756	0.04	2.98

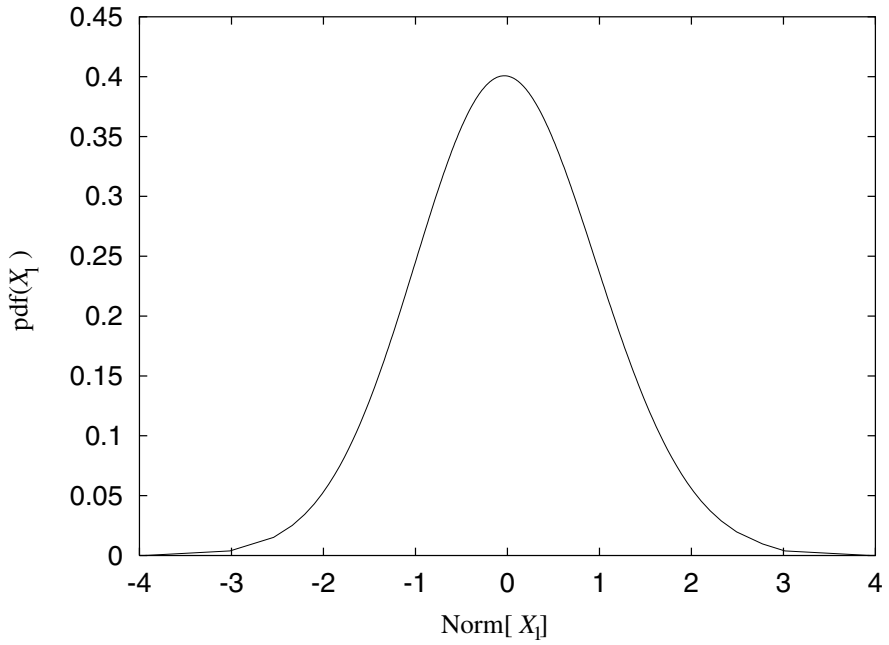

Figure 5.19: Normalized pdf(X_1) for NAS-EP.

As the moments of X are available, the moments of Y can be predicted using Eq. (5.18) and compared to the measured value of Y using Eq. (5.43). As shown in Figure 5.20, indeed the small error is consistent with Figure 5.10 for the standard normal distribution. The difference between both figures is due to the largely differing coefficient of variance between the two workloads, and also due to measurement inaccuracies. Figure 5.20 also shows the error of Eq. (5.44) which decreases for large N. For N up to 128 our method is better than that in Eq. (5.44).

5.6.2 SSSP

In this experiment we model a simple parallel version of single-source shortest-path (SSSP) problem as described in [82]. The input is a $V \times V$ matrix representing a weighted,

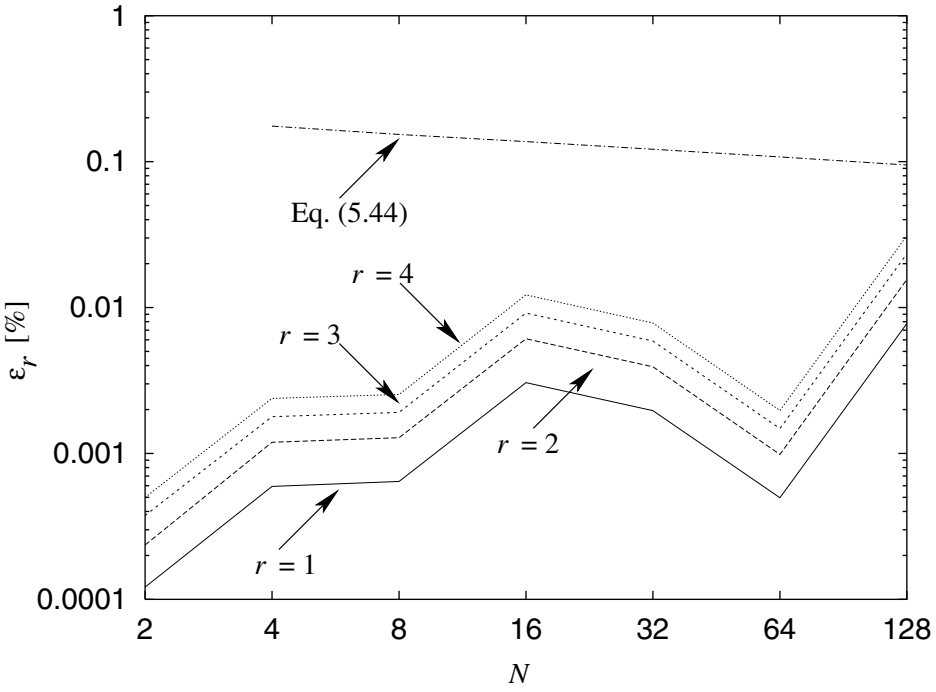

Figure 5.20: ε_r [%] for NAS-EP.

directed graph with V vertices, while the output of SSSP is a vector denoting the shortest distance between the source vertex to all other vertices. The weight matrix is assigned real values, uniformly distributed in the interval (0,10), while infinite weights are assigned with probability 0.1. As workload X_i we count the floating point operations per task. The moments of X_1 are presented in Table 5.4. The pdf of normalized, left-skewed X_1 for $V = 1,000$ is given in Figure 5.21.

To evaluate N-ary and-parallel composition we let N independent SSSP programs run in parallel. Each computes the vector for a different vertex, referred to as APSP, all-pairs shortest path, performing SSSP for all nodes. Figure 5.22 shows that ε_r[%] is less than 1 % where N ranges from 2 to 128. We also compared our approach with that of Gumbel in Eq. (5.44). In contrast to the prediction error in Figure 5.20 the ε_1 for the Gumbel approach now increases for large N due to the left-skewed X_1.

To evaluate binary parallel composition we have parallelized SSSP, based on an *internal* farmer/worker organization. As a result, correlation of the two tasks is to be expected. The corresponding ε_r[%] for various V is given in Figure 5.23 where V ranges from 100 to 1,000. Indeed, due to the correlation effect ε_r[%] for binary parallel composition is

Table 5.4: The moments of X_1 for SSSP

$\mathsf{E}[X_1]$	$\mathsf{Var}[X_1]$	$\mathsf{Skw}[X_1]$	$\mathsf{Kur}[X_1]$
$4.7\ 10^6$	$1.7\ 10^{11}$	0.48	3.34

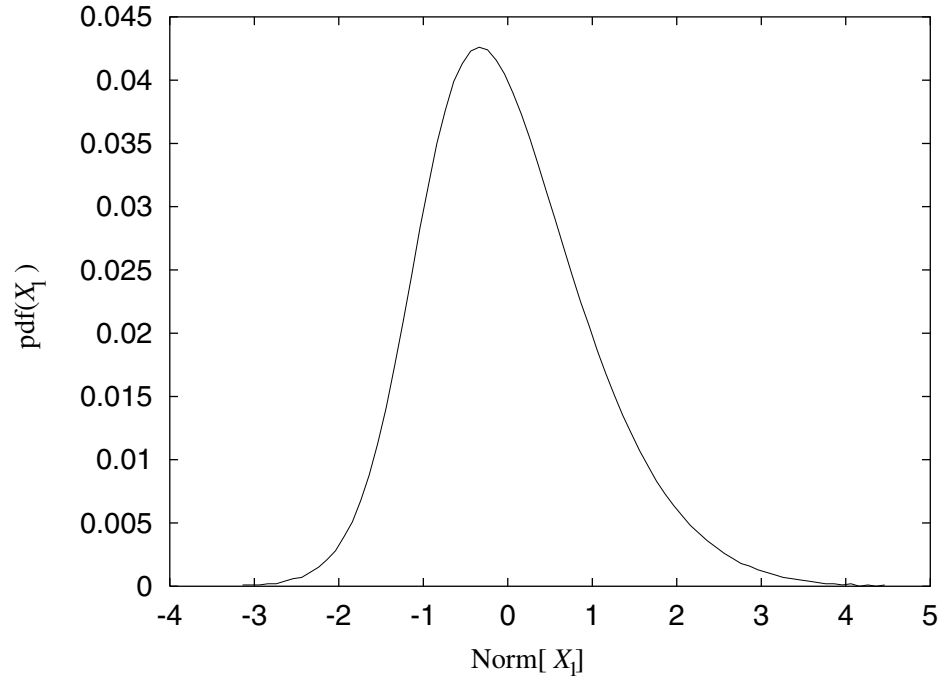

Figure 5.21: Normalized pdf(X_1) for SSSP

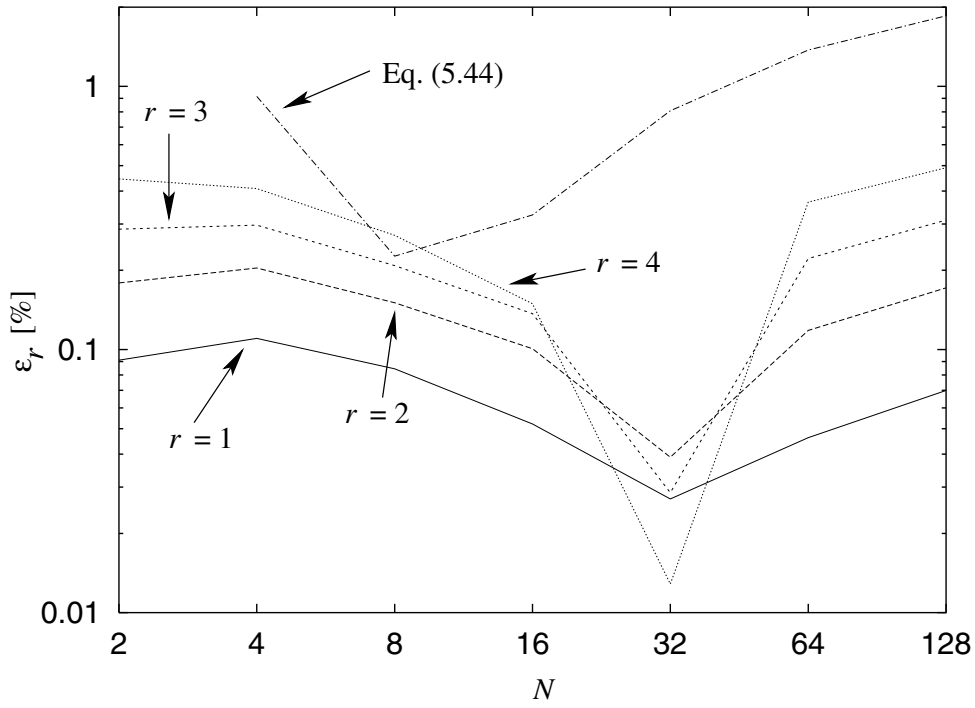

Figure 5.22: $\varepsilon_r[\%]$ for APSP

somewhat larger than that for N-ary parallel composition, but is insensitive to V value variation.

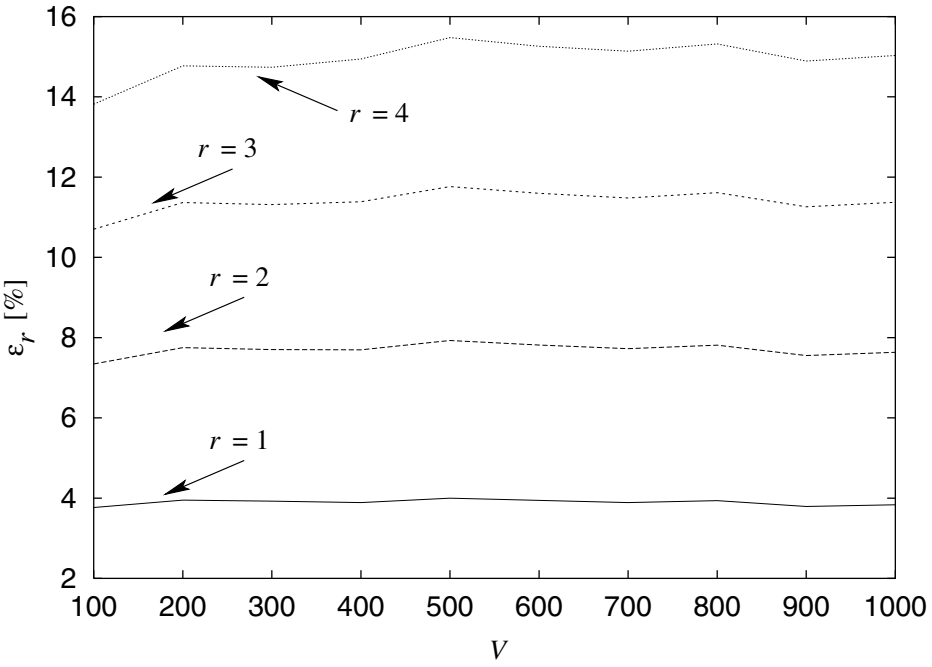

Figure 5.23: $\varepsilon_r[\%]$ for SSSP

5.6.3 PSRS

In this experiment we apply our prediction method to the parallel sorting by regular sampling (PSRS) algorithm. The algorithm features three (data) parallel sections. The first parallel section, called `sort`, divides the array in P equal subarrays. Each partition is sorted in parallel, after which a number of global pivots is determined. Based on these pivots, the second parallel section, called `disjoin`, divides the subarray in P parts. The third section, called `merge`, merges the subarrays cyclically with regard to the processor index. As a result, a sorted array is obtained. A description of the algorithm can be found in [97].

The working of `sort` implies that each task has an iid workload (uncorrelated), since we generate the input array with a uniform random variable using sample space $[0, 1]$. In contrast, the tasks in the other two parallel sections are correlated, since their workload depends on the same global pivot values. Again, as workload X_p we choose the number of floating point operations per task.

In our experiments we keep the average workload of each task constant (1,000 array elements per task), while N ranges between 2 and 128 (measurements for larger N values prohibited by current run time systems limitations). Table 5.5 shows the central moments of the execution times X_1 for $N = 2$ for 1,000 sample data sets. The pdf of X_1 is given

in Figure 5.24, where the mean and variance are normalized. Clearly, the distributions of X are quite different from any of the standard distributions we have examined in the previous section.

Table 5.5: The moments of X_1 for PSRS

Sections	$\mathsf{E}[X_1]$	$\mathsf{Var}[X_1]$	$\mathsf{Skw}[X_1]$	$\mathsf{Kur}[X_1]$
`sort`	9,581	$1.04\ 10^5$	0.94	4.31
`disjoin`	496	177	-1.64	5.58
`merge`	1,475	545	-1.54	6.06

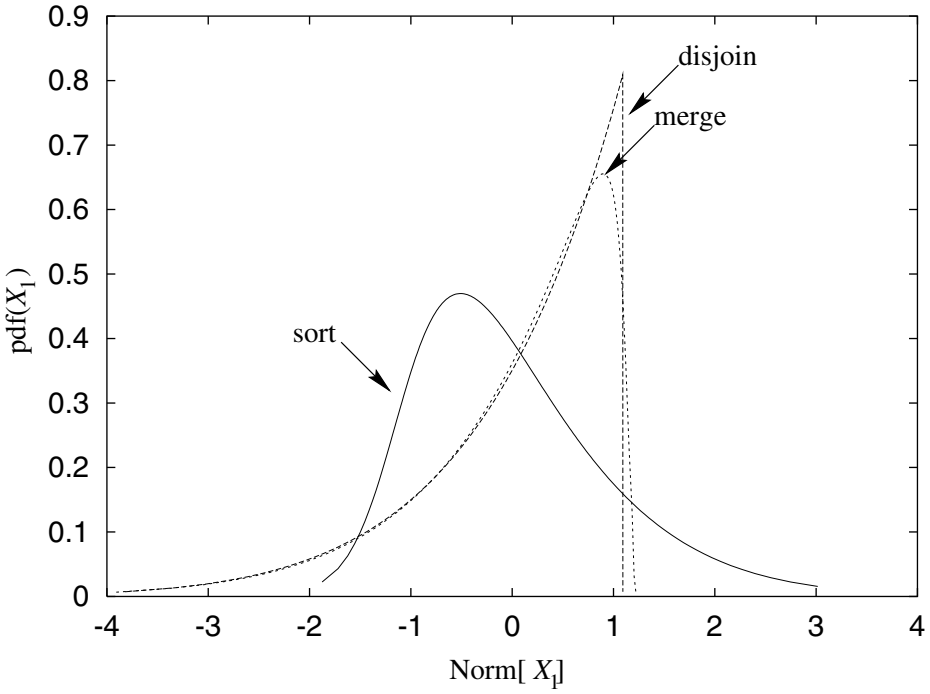

Figure 5.24: Normalized pdf(X_1) for PSRS

The ε_r for the three parallel sections are given in Figures 5.25, 5.26, and 5.27, respectively. As the tasks in the first section are independent and have equal workload, the iid requirement applies, which permits Eq. (5.18) to be used. Figure 5.25 shows that the error is less than 1% for $N \leq 128$ (cf. Figure 5.10). Figure 5.26 indicates that Eq. (5.18) still provides a good approximation although the iid assumption does not quite hold.

Figure 5.27 shows that the relative error is much larger than that in Figures 5.25 and 5.26. Clearly, our approach is inadequate to predict $\mathsf{E}[Y^r]$ when X_i are highly correlated. Correlation data for the `merge` parallel section is shown in Figure 5.28 in terms of coefficient of correlation between P_1 and P_j for $P = 16$. Despite the small coefficient of correlation the actual covariance is large due to large variance values.

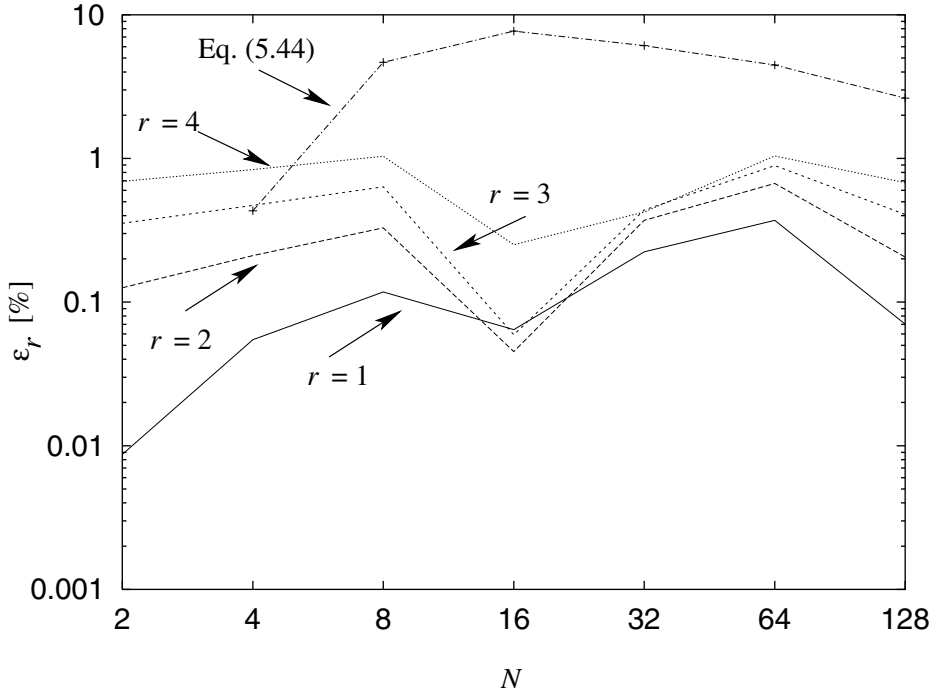

Figure 5.25: $\varepsilon_r[\%]$ for `sort`

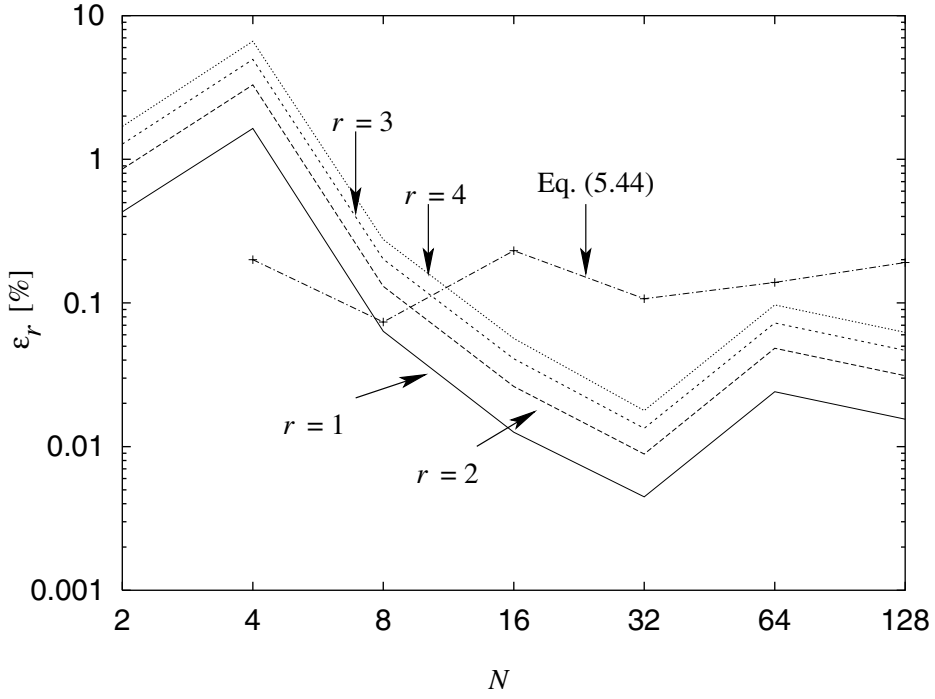

Figure 5.26: $\varepsilon_r[\%]$ for `disjoin`

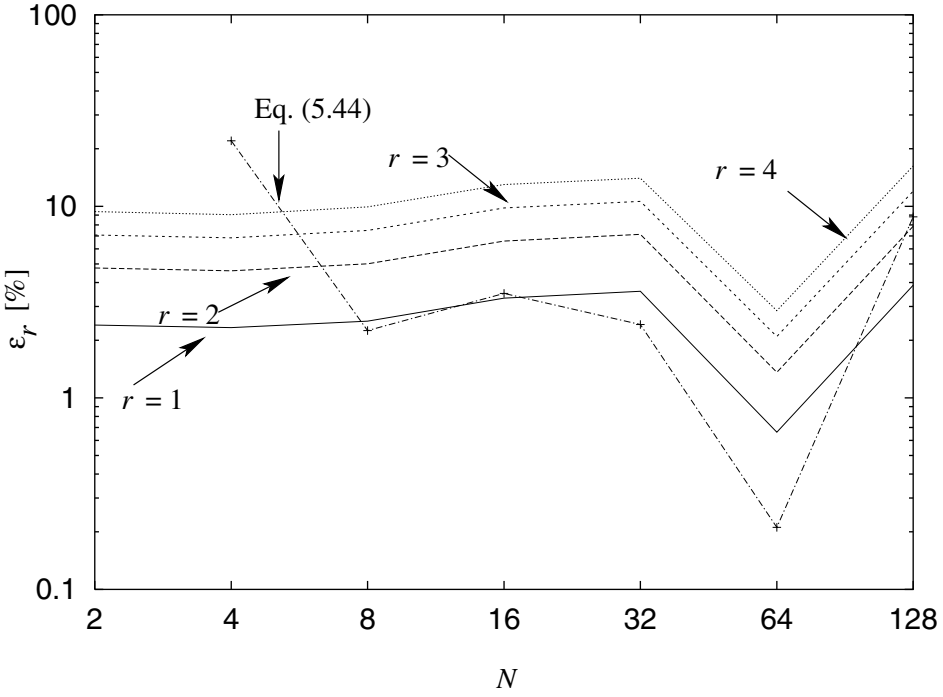

Figure 5.27: $\varepsilon_r[\%]$ for `merge`

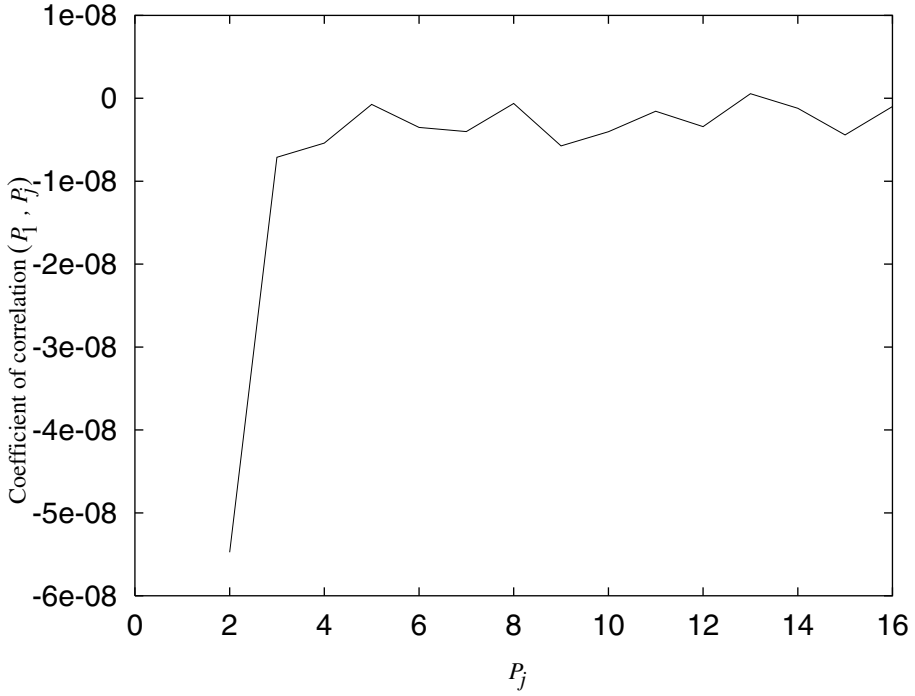

Figure 5.28: Coefficient of correlation for `merge`

Figures 5.25, 5.26, and 5.27 also show the error using Gumbel's prediction in Eq. (5.44). Since Eq. (5.44) does not apply for $N = 2$, the error is computed for $N = 4$ to $N = 128$. Figures 5.25 and 5.26 show that Eq. (5.18) is better than that of Gumbel in Eq. (5.44), which only applies to symmetric distributions. Since the third parallel section is highly correlated, the prediction error of both methods is comparable as shown by Figure 5.27.

5.6.4 WATOR

WATOR is a Monte Carlo simulation in which idealized fishes and sharks live, move randomly, breed, and eat one another in a two-dimensional ocean with toroidal topology [4]. The characteristic of such an algorithm is that the workload can be severally unbalanced because of computational scenario. Hence, the workload in each processor is changing with the increasing simulated time t. The load inbalance comes about naturally because of the dynamics of the problem: the fishes and sharks tend to aggregate in schools as they breed, move, and eat each other. Figure 5.29 shows an initial condition of the fishes ("+") and sharks ("×") in the ocean with 10,000 grid points. They agregate in schools gradually as shown in Figures 5.30 to 5.32.

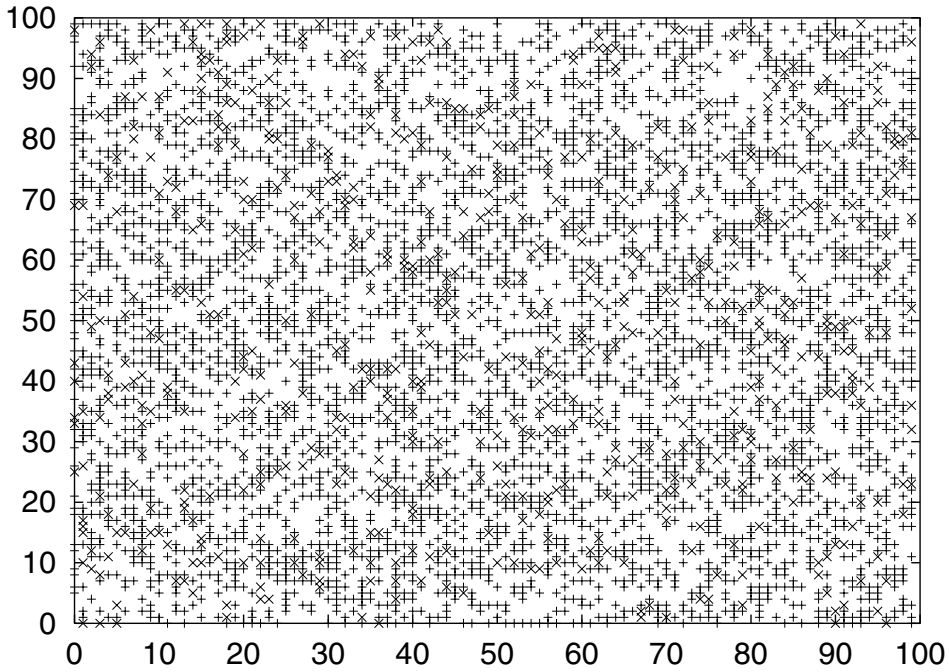

Figure 5.29: Snapshot of WATOR for $t = 0$

In our experiment, we perform a simple rectangular subdomain decomposition of the ocean such that each processor is assigned to process 2,500 grid points. As workload X_i, (X_1 shown in Table 5.6), we choose the number of fish within each processor which on initialization is generated randomly over each location in the ocean. The moments in the table are based on 6,000 simulation runs. The pdf of workload X_1, where X_1 is

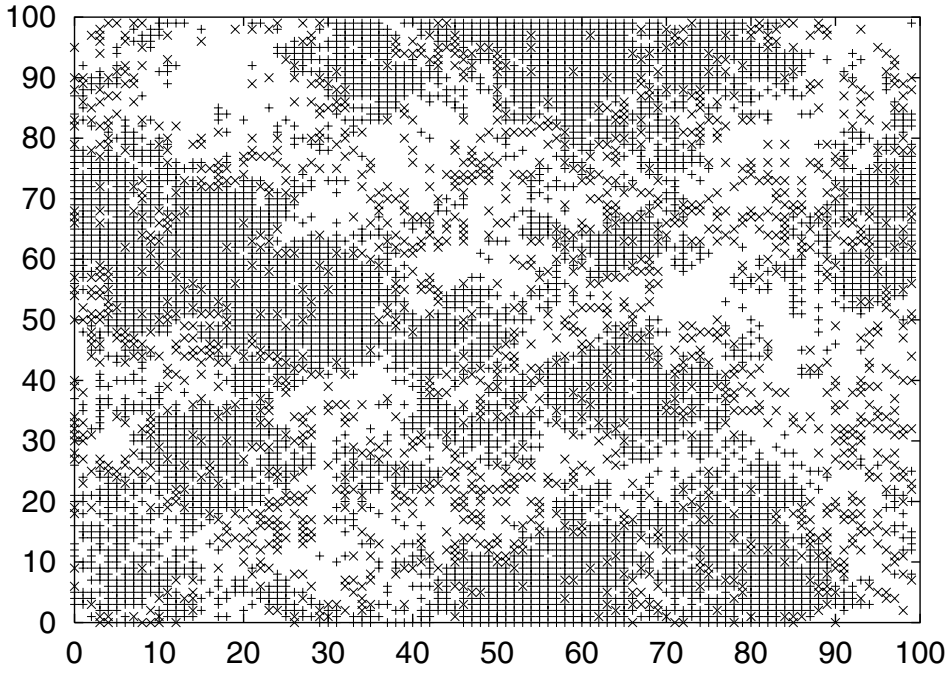

Figure 5.30: Snapshot of WATOR for $t = 16$

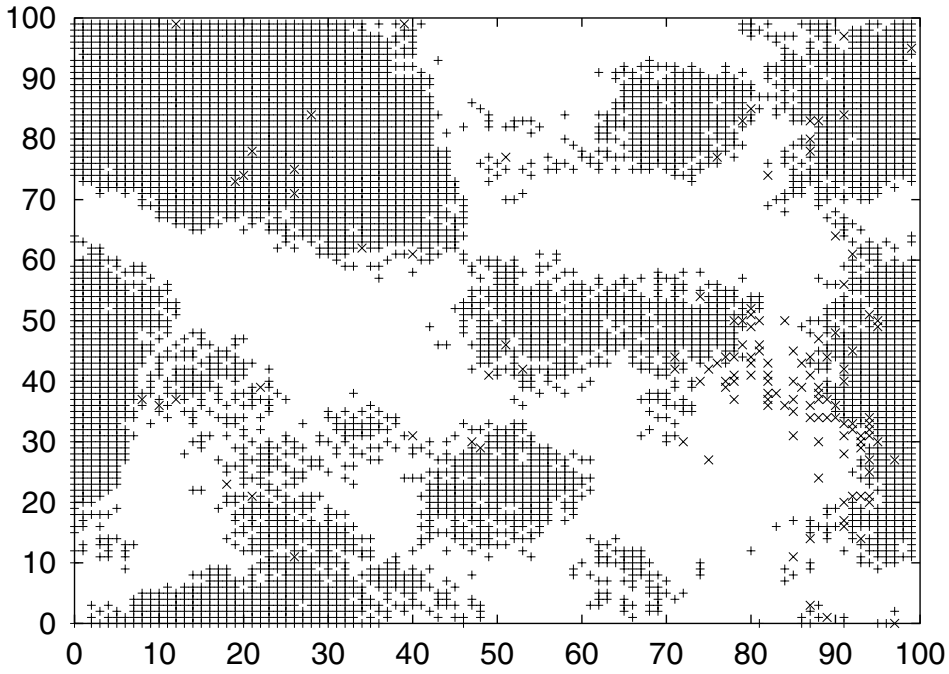

Figure 5.31: Snapshot of WATOR for $t = 48$

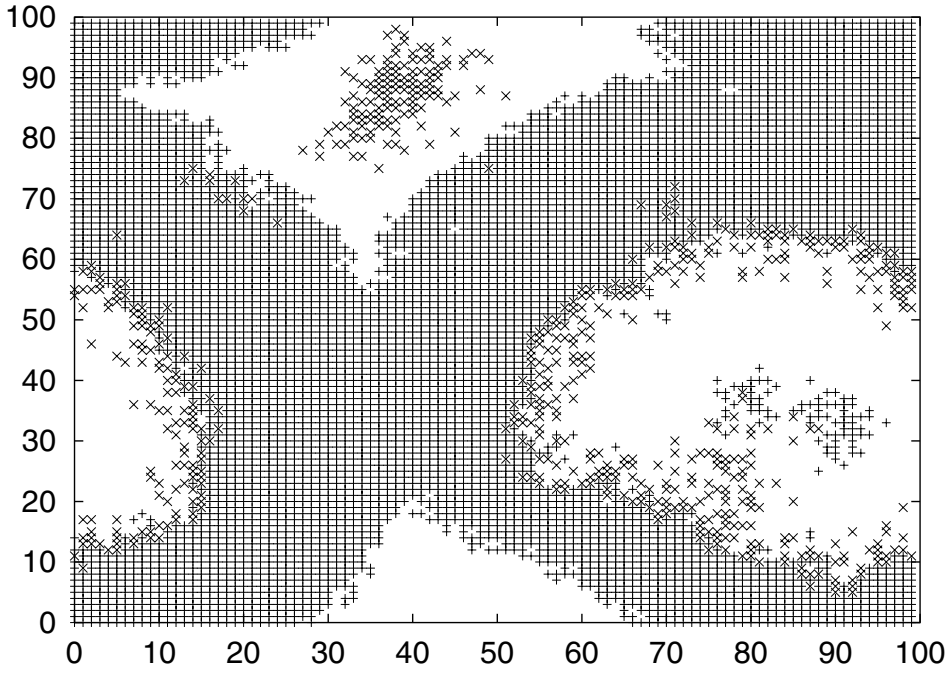

Figure 5.32: Snapshot of WATOR for $t = 128$

normalized, is also shown in Figure 5.33. As shown in this figure, the workload exhibits a bimodal distribution for $t = 256$.

Table 5.6: The moments for WATOR

t	$\mathsf{E}[X_1]$	$\mathsf{Var}[X_1]$	$\mathsf{Skw}[X_1]$	$\mathsf{Kur}[X_1]$
0	1,000	594	−0.02	2.99
4	491	175	0.07	3.01
16	895	7,421	0.36	2.80
64	1,762	208,695	−0.40	2.50
128	1,278	230,641	1.02	3.32
256	1,487	495,116	0.22	1.86

Since in WATOR the slowest processor determines the performance, we apply X_i to N-ary and-parallel composition using Eq. (5.18). The ε_r for $t = 0, 4, 16, 64, 128$, and 256 are presented in Figures 5.34, 5.35, 5.36, 5.37, 5.38 and 5.39, respectively. We vary N from 2 to 128. The smallest ε_r appears in Figure 5.34 for $t = 0$ since X_i are still iid. In the figure, ε_r increases logarithmically for large N (cf. Figure 5.10). The error increases with increasing t due to higher correlation between the tasks caused by clustering of workload within certain processors as shown in Figures 5.29 to 5.32 which displays the grid for $t = 0, 16, 48$ and 128, respectively. In these figures, the toroidal topology has 100×100 grids.

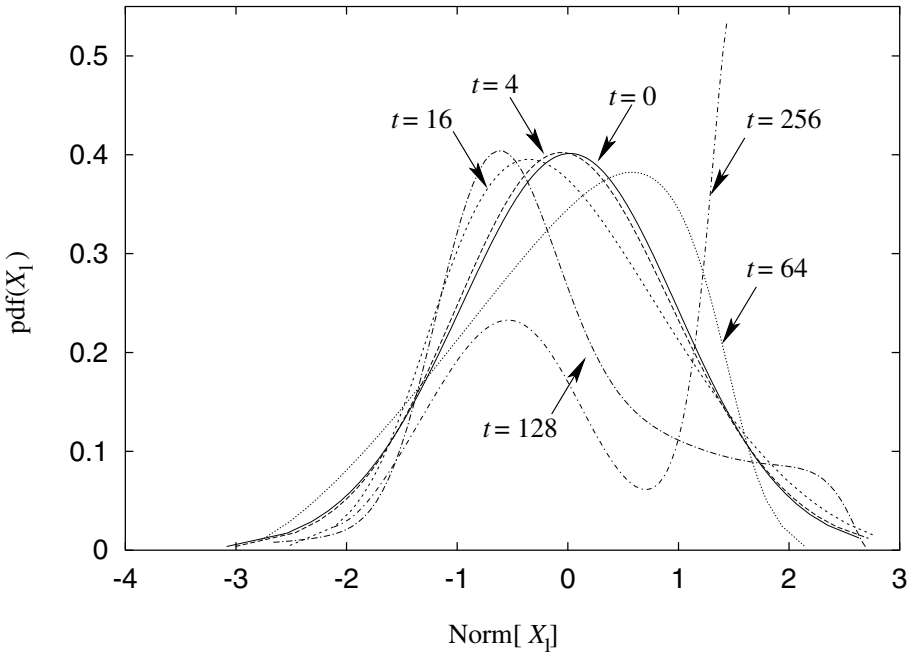

Figure 5.33: Normalized pdf(X_1) for WATOR

Figure 5.40 summarizes the correlation between P_1 and P_j for $P = 16$ processors. Again, despite the small coefficient of correlation the actual covariance is large due to large variance values. As to be expected, Figure 5.40 shows that the correlation is the largest for the processors nearest to P_1 (P_2 and P_{16}, the processors are interconnected in a 1-D torus). Figure 5.40 also shows that the correlation increases with t.

Apart from correlation effects we observe a bimodal distribution for X_1 ($t = 256$). Such variability of distribution strongly influences the accuracy of our approach since our (GLD) analysis applies to unimodal distributions only. Consequently, for workloads with multi-modal distributions a different approach needs to be taken. A method that deals with multi-modal workloads is presented in [62] for queuing network models.

5.6.5 Pipeline

In this section we determine the quality of our approximation when applied to a pipelined application of two tasks whose execution times are denoted by X_1 and X_2, respectively. The pipeline comprises a Gaussian random generator task that supplies one-dimensional, real-valued vectors of length N to a PSRS sorting task. In steady state the total execution time per vector of the pipelined application is given by $Y = \max(X_1, X_2)$, assuming both tasks use different resources. The generator task is implemented using the data parallel NAS-EP benchmark [8]. The PSRS sorting task has been described in Section 5.6.3.

To have an interesting scenario for our experiment (i.e., the worst case for Eqs. (5.22) and (5.21)), we adjust the parameters such that both tasks have approximately an equal

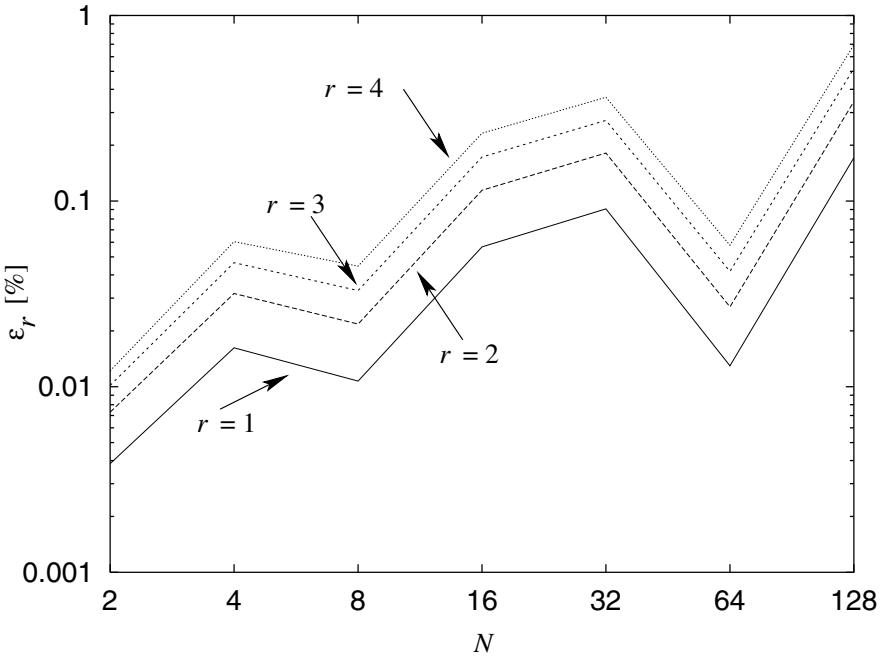

Figure 5.34: ε_r for WATOR ($t = 0$)

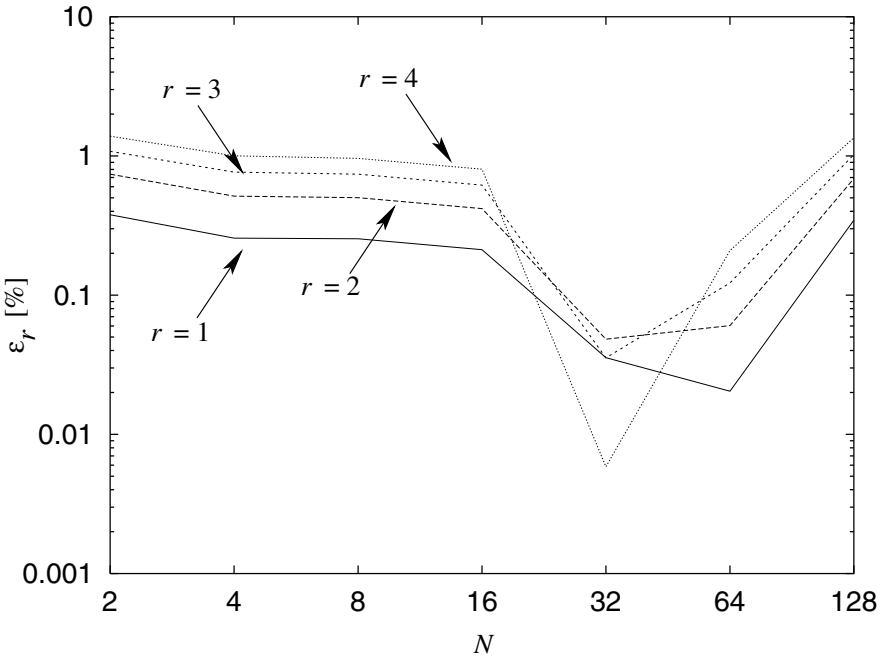

Figure 5.35: ε_r for WATOR ($t = 4$)

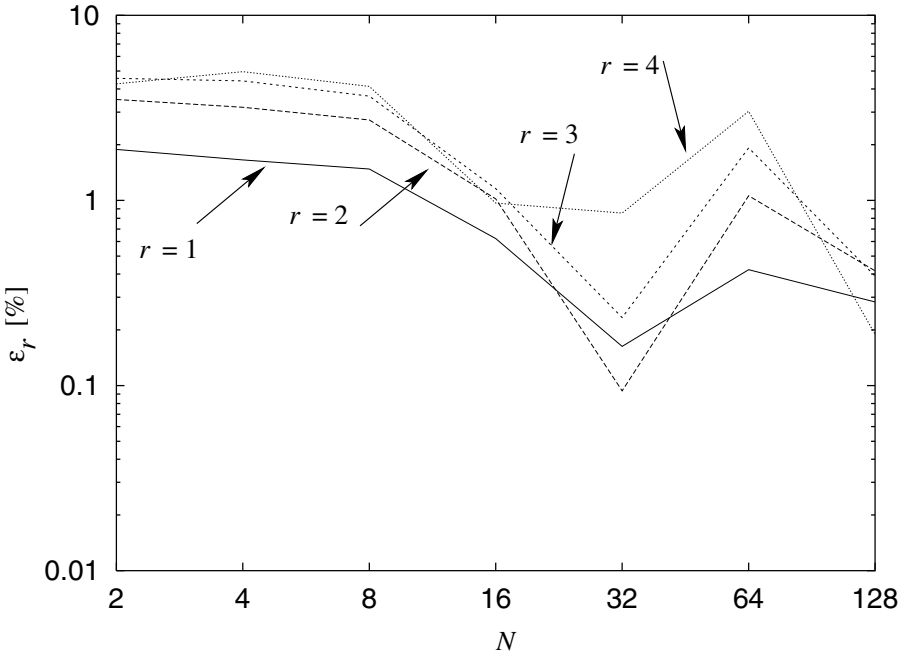

Figure 5.36: ε_r for WATOR ($t = 16$)

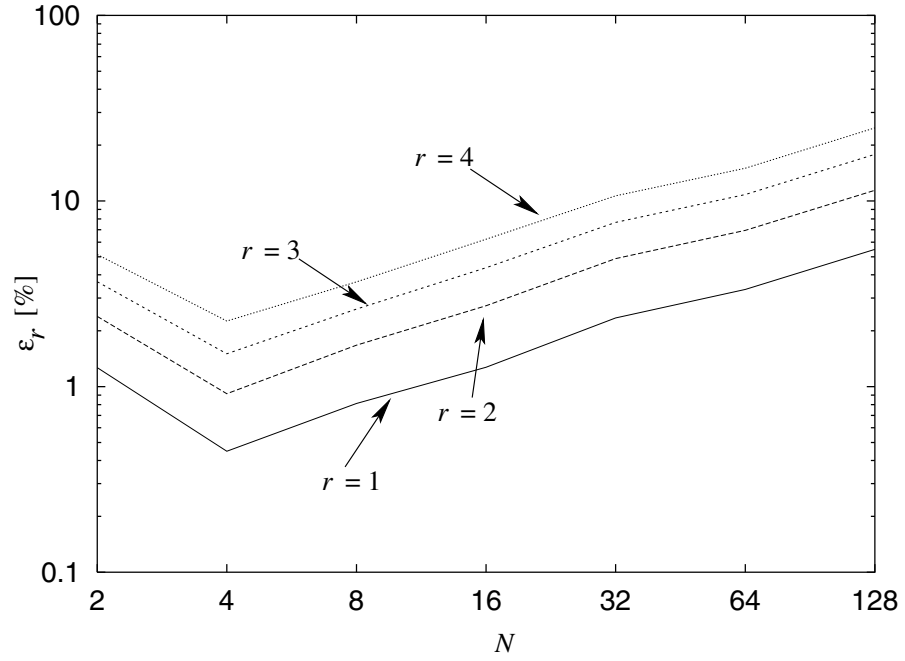

Figure 5.37: ε_r for WATOR ($t = 64$)

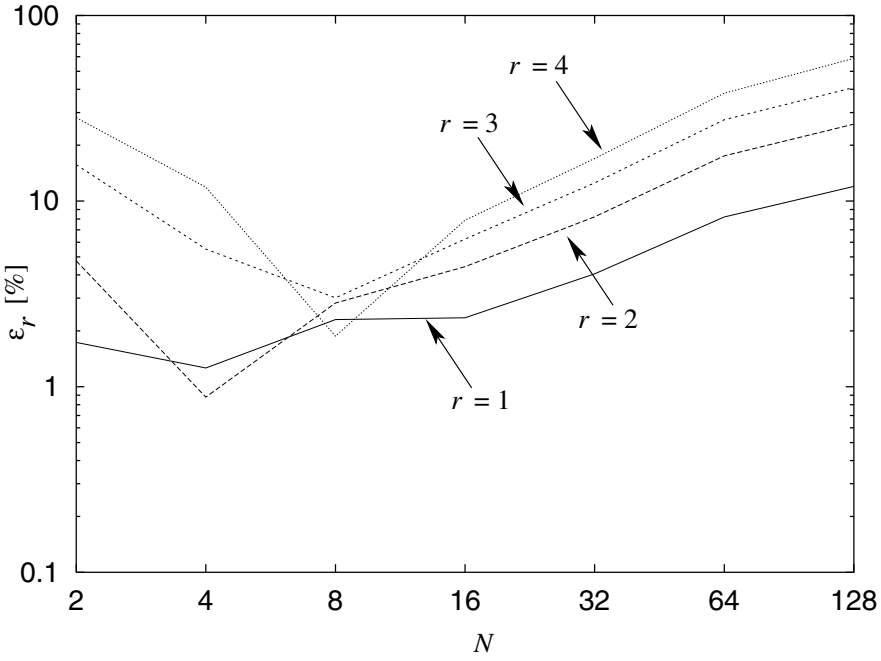

Figure 5.38: ε_r for WATOR (t=128)

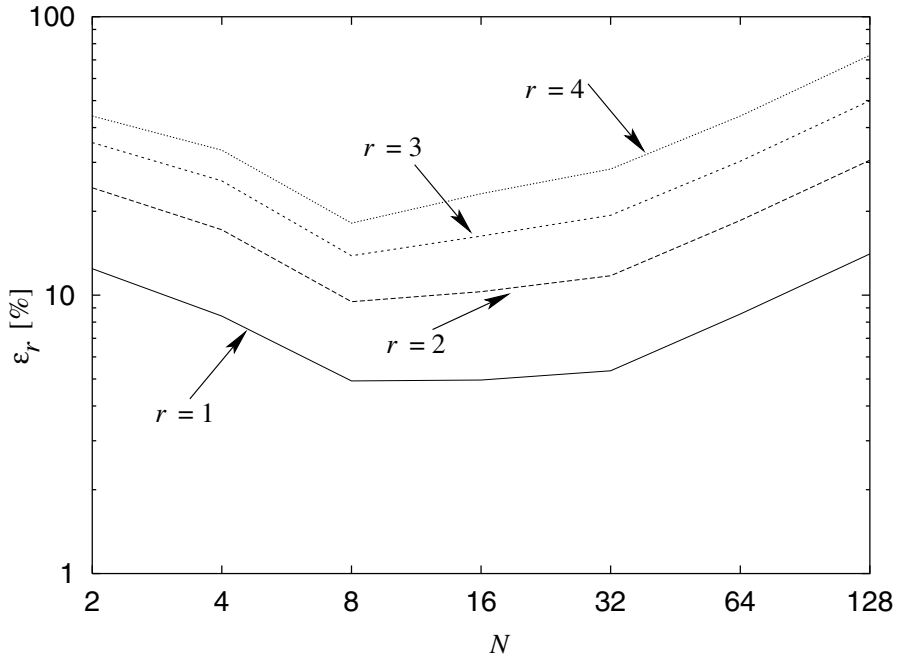

Figure 5.39: ε_r for WATOR ($t = 256$)

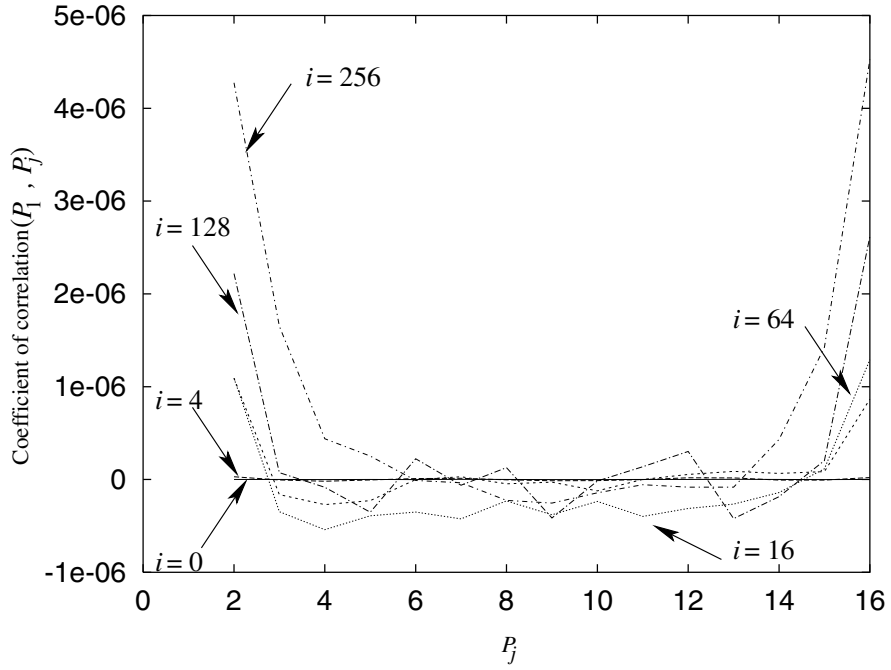

Figure 5.40: Coefficient of correlation for WATOR

workload (which also balances the pipeline). Hence, we choose $P = 1$ processor for NAS-EP and $P = 24$ for PSRS. The pipeline processes $6,000$ arrays with length N per experiment. To verify the sensitivity of prediction accuracy on N, we perform a number of experiments where N varies from $N = 4 \cdot 10^4$ to $N = 10^5$. The execution times of X_1 and X_2 are given in Tables 5.7 in terms of the first four moments, based on $6,000$ simulation runs. The pdfs of X_1 and X_2 are shown in Figure 5.41 for $N = 10^5$. In this figure X_1 is approximately normal while X_2 has a long right tail. Furthermore, the variance of X_2 (the sorting task) is much larger than that of X_1.

Table 5.7: The first four moments of X_1 and X_2

N	$\mathsf{E}[X_1]$	$\mathsf{Var}[X_1]$	$\mathsf{Skw}[X_1]$	$\mathsf{Kur}[X_1]$	$\mathsf{E}[X_2]$	$\mathsf{Var}[X_2]$	$\mathsf{Skw}[X_2]$	$\mathsf{Kur}[X_2]$
$4 \cdot 10^4$	$4.55\ 10^4$	$6.78\ 10^3$	0.17	2.90	$4.57\ 10^4$	$5.52\ 10^5$	0.74	4.09
$5 \cdot 10^4$	$5.68\ 10^4$	$8.55\ 10^3$	-0.07	3.11	$5.64\ 10^4$	$7.92\ 10^5$	0.36	2.81
$6 \cdot 10^4$	$6.82\ 10^4$	$1.06\ 10^3$	0.04	2.79	$6.75\ 10^4$	$1.17\ 10^6$	0.74	4.32
$7 \cdot 10^4$	$7.96\ 10^4$	$1.19\ 10^4$	0.06	3.05	$7.84\ 10^4$	$1.25\ 10^6$	1.17	6.18
$8 \cdot 10^4$	$9.09\ 10^4$	$1.39\ 10^4$	0.03	3.12	$8.95\ 10^4$	$1.68\ 10^6$	0.64	3.53
$9 \cdot 10^4$	$1.02\ 10^5$	$1.69\ 10^4$	-0.02	3.02	$1.01\ 10^5$	$1.73\ 10^6$	0.73	3.89
$1 \cdot 10^5$	$1.14\ 10^5$	$1.73\ 10^4$	-0.04	2.93	$1.12\ 10^5$	$2.15\ 10^6$	0.73	4.25

We apply the moments of X_1 and X_2 in the binary and-parallel composition based on

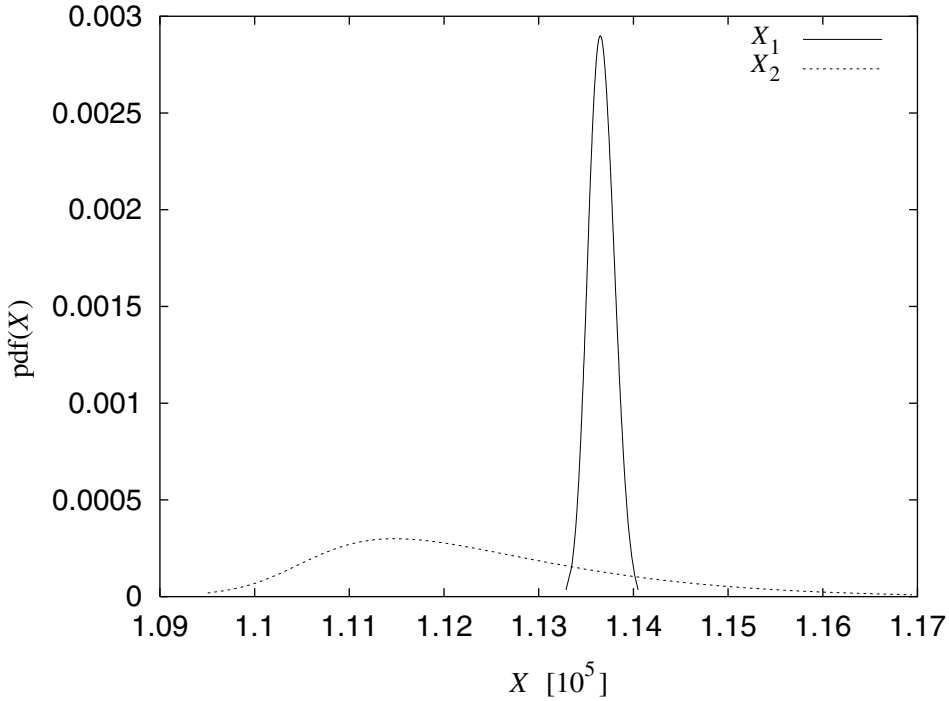

Figure 5.41: pdf(X_1) and pdf(X_2) for $N = 10^5$

Newton's method and the heuristic in Eqs. (5.35) and (5.21), respectively. Figure 5.42 shows ε_r of Newton's method and the heuristic. For both methods ε_r is decreasing for large N due to the disjunction of X_1 and X_2. Although $\mathsf{E}[X_1] = E[X_2]$, ε_r is excellent since the coefficient of variance is small (in practice, task variance is indeed much smaller than we have assumed in our analysis in Section 5.5).

5.6.6 Parallel Search

While the previous distributions already provide an indication of the performance of our technique, in this section we present the results for empirical distributions as measured from a distributed search application (using search engines) on the internet. For a large number of queries, each query is sent to each site in parallel. Since the response time is determined by the fastest response, this experiment constitutes or-parallel composition. The search engines are located at URL addresses in the USA (dot com), Europe (dot nl), and Latin America (dot cl). As query we search 2,000 large cities around the world while as response we obtain html text.

In order to obtain more workload distributions, in our experiment, we distinguish two workload distributions associated with input distribution X, i.e., the response time T_r and the search time T_s (both in seconds). T_r denotes the time needed for sending a query, searching, and receiving the response from the search engines while T_s presents only the time needed for a search engine to find the query excluding the round trip communication delay. Although both workloads are measured from the same experiment, the distribution of T_s has different shape from T_r.

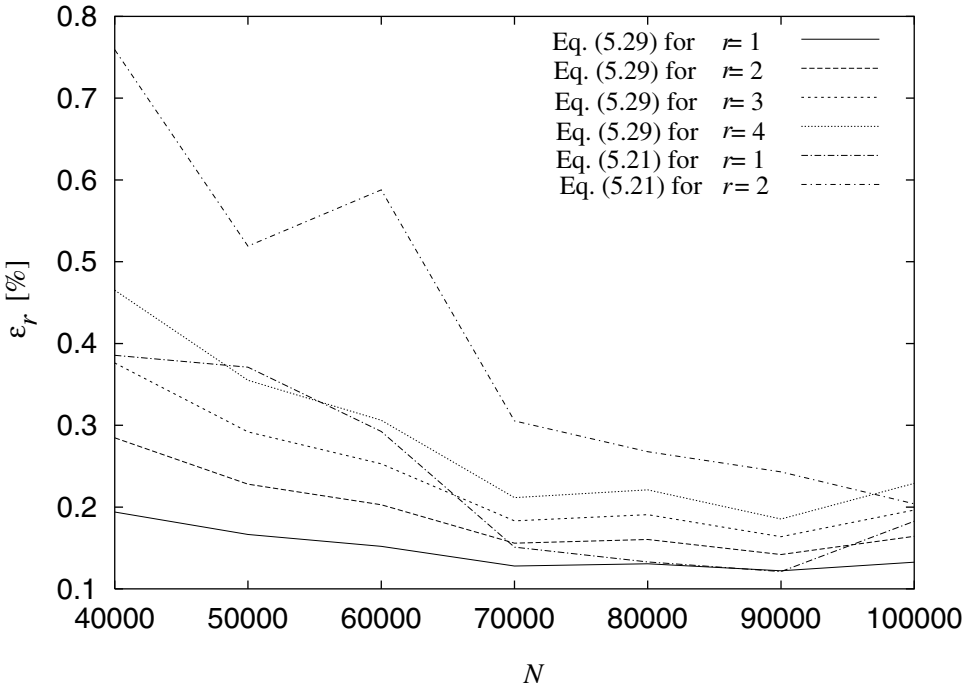

Figure 5.42: ε_r for pipeline

In an N-ary or-parallel search, we represent all sites on one particular domain in terms of one workload distribution X and apply Eq. (5.20). To resemble identical distributions for X we repeat the search on one domain up to 128 times in unconsecutive manner to avoid probable caching mechanism in the search engine. For $X = T_r$ the distributions are shown in Figure 5.43. All distributions are unimodal and slightly left-skewed. The corresponding moment error for N-ary or-parallel composition is shown in Figures 5.44, 5.45 and 5.46 for the dot cl, dot com, and dot nl domains, respectively. The errors are moderate while the mean errors are in the few percent range. In general, the errors increase slowly for large N.

The distributions of T_s are shown in Figure 5.47. The dot cl domain has a left-skewed distribution while the dot com and dot nl domains have bimodal distributions. For all three distributions the left tails are much sharper in comparison with those of T_r. The corresponding prediction error for the N-ary or-parallel composition is shown in Figures 5.48, 5.49 and 5.50 for the dot cl, dot com and dot nl domains, respectively. For all three cases ε_r increases for larger N. The average error for the dot cl domain is appreciable since the distribution is sharply left-skewed where the distribution mass is concentrated near the minimum value. In such case the fitting for lambda values is very sensitive. The error for the dot com and dot nl domains are much worse and increase rapidly for larger N, which is caused by the fact that the distribution of X is no longer unimodal, a requirement of our technique. Another error source is the small number of discrete points for X. Consequently, for N large the minimum of X will be deterministic.

To study a binary or-parallel search we combine each possible combination of two domains and apply Eq. (5.42). For $X = T_r$ the prediction error for all three cases is in the

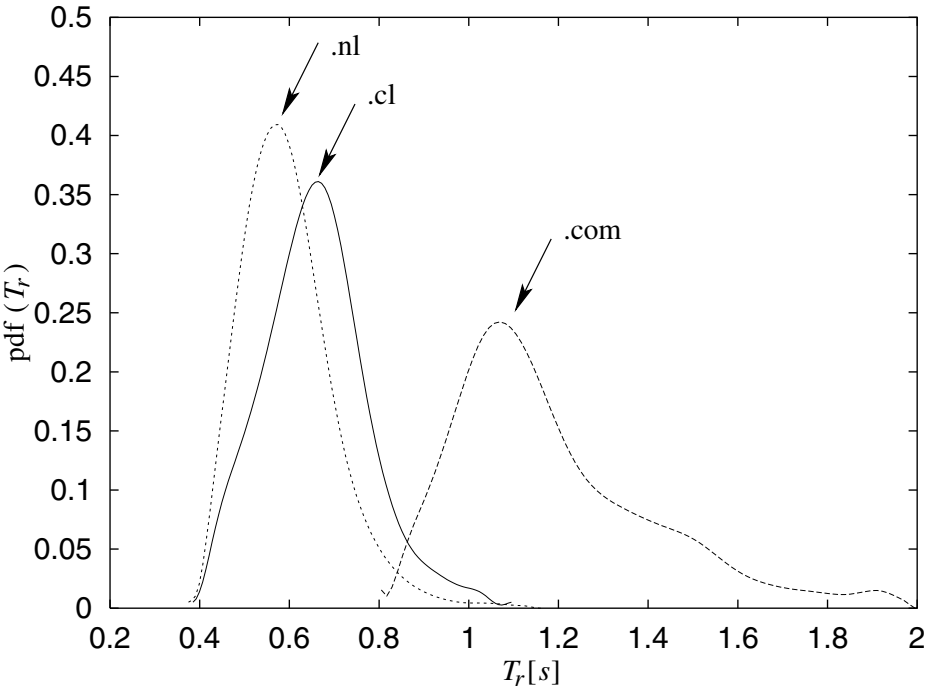

Figure 5.43: The pdf of T_r

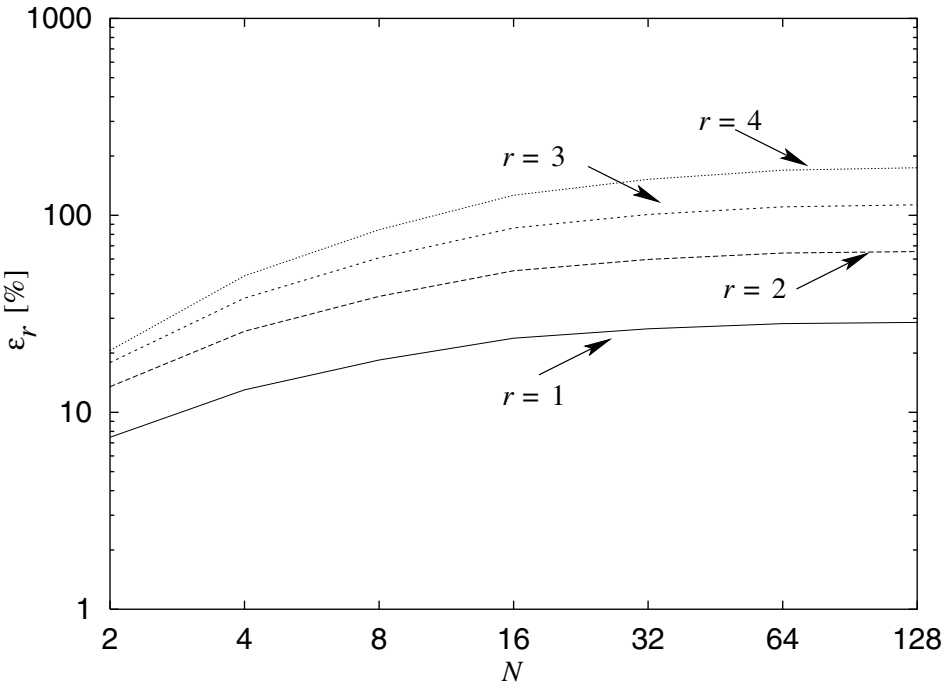

Figure 5.44: $\varepsilon_r[\%]$ for N-ary or-parallel using T_r (.cl)

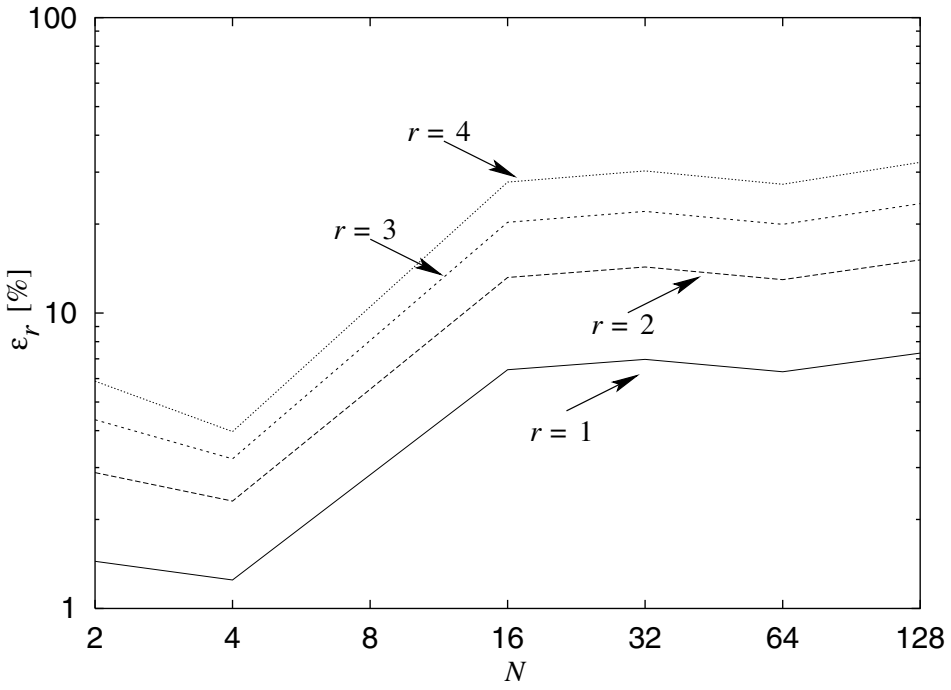

Figure 5.45: ε_r[%] for N-ary or-parallel using T_r (.com)

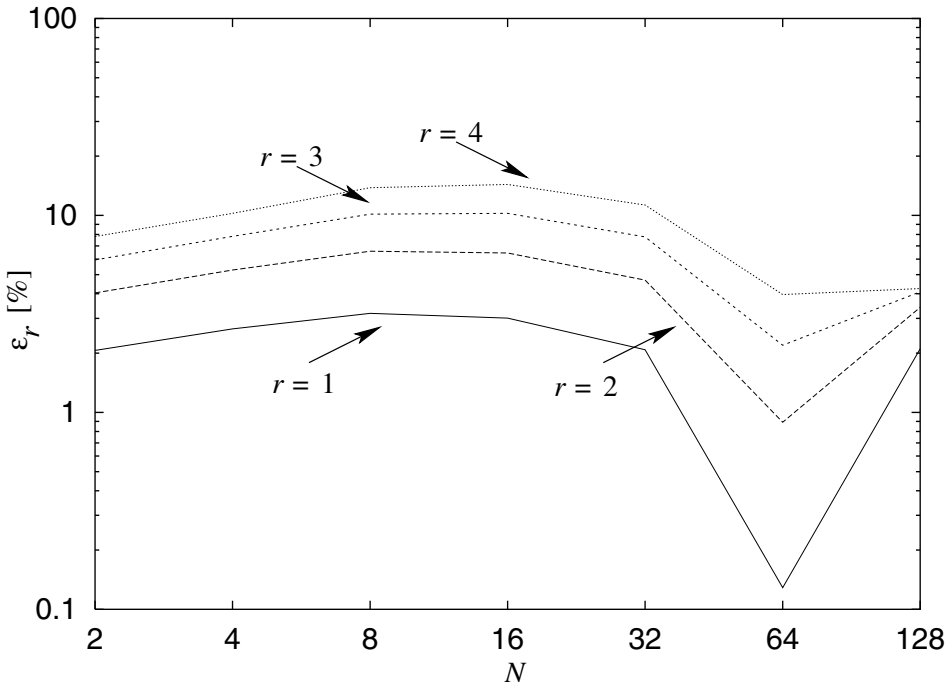

Figure 5.46: ε_r[%] for N-ary or-parallel using T_r (.nl)

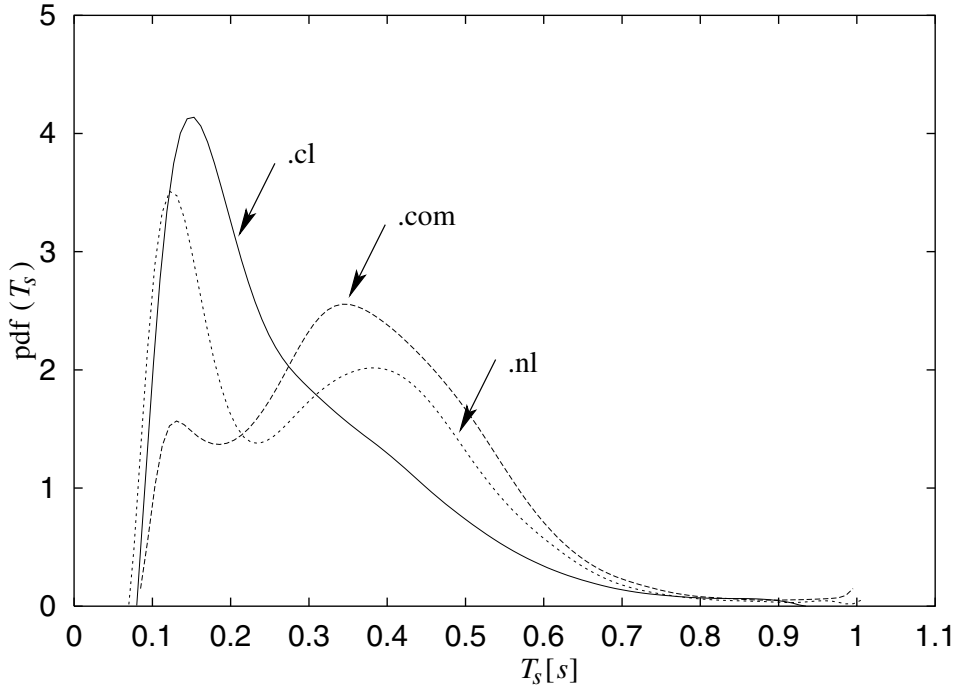

Figure 5.47: The pdf of T_s

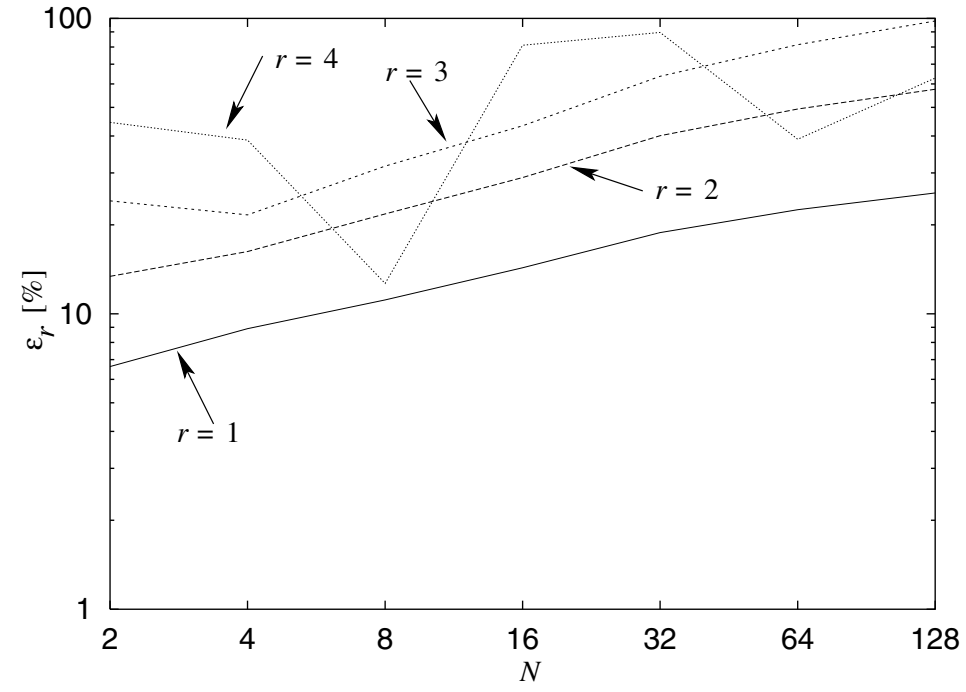

Figure 5.48: $\varepsilon_r[\%]$ for N-ary or-parallel using T_s (.cl)

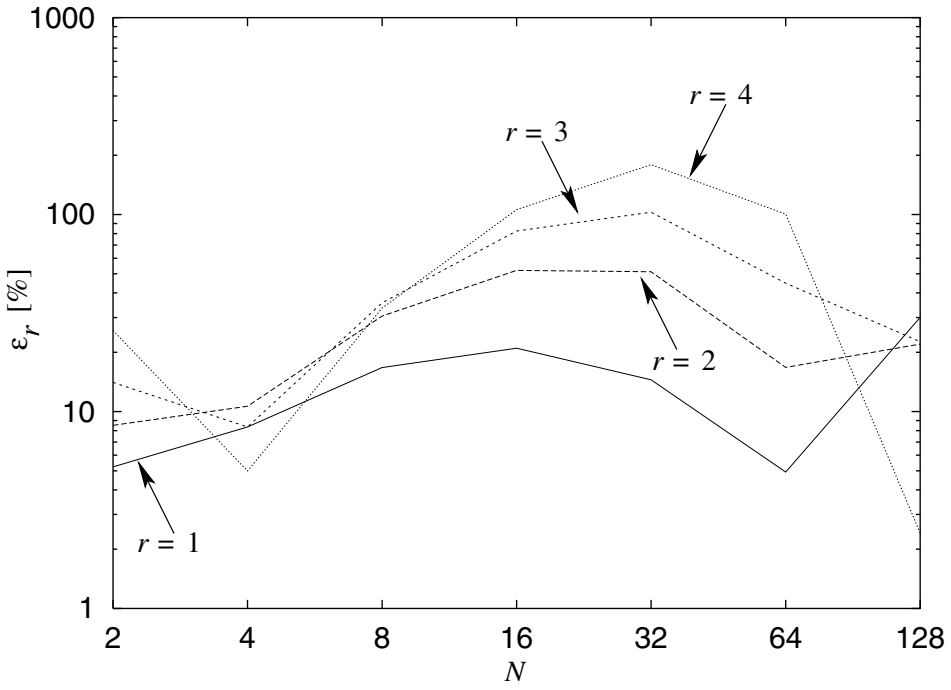

Figure 5.49: ε_r[%] for N-ary or-parallel using T_s (.com)

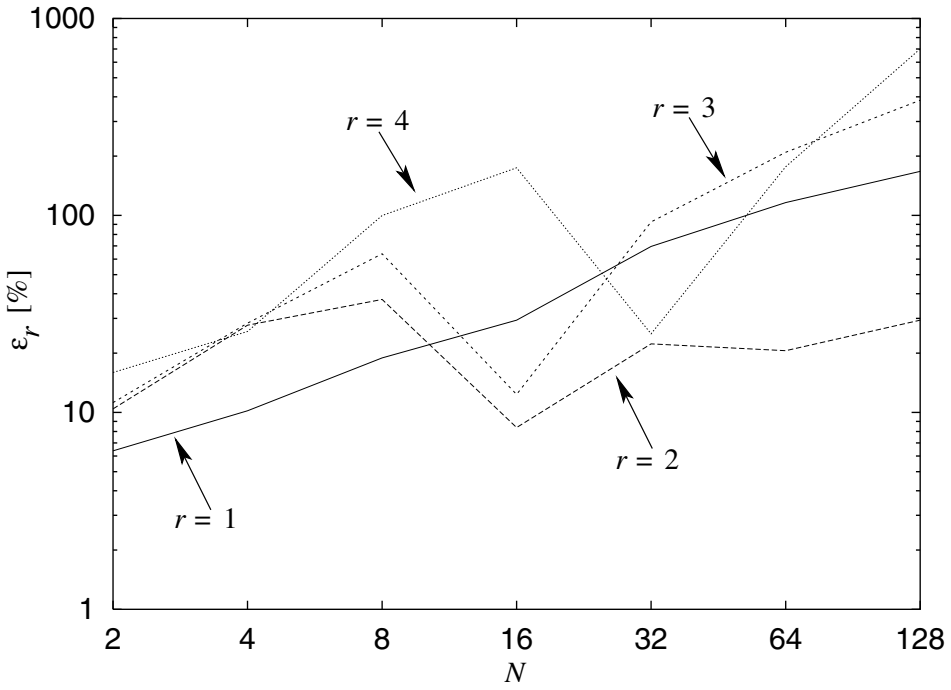

Figure 5.50: ε_r[%] for N-ary or-parallel using T_s (.nl)

few percent range as shown in Table 5.8 (left) since again, the workloads are unimodal. For $X = T_s$ the prediction error for the mean is in the few percent range as given in Table 5.8 (right) while for the higher moments the prediction error is moderate, since again, the workloads are no longer unimodal.

Table 5.8: $\varepsilon_r[\%]$ for binary Or-parallel (T_r: left and T_s: right)

$T_{r,1}$	$T_{r,2}$	ε_1	ε_2	ε_3	ε_4
.cl	.com	0.2	0.5	1.0	1.6
.com	.nl	0.1	0.4	1.1	2.5
.nl	.cl	0.3	0.9	1.8	3.5

$T_{s,1}$	$T_{s,2}$	ε_1	ε_2	ε_3	ε_4
.cl	.com	4.0	8.5	21	73
.com	.nl	1.1	9.3	21	45
.nl	.cl	1.8	4.7	19	37

5.7 Summary

In this chapter we have presented an analytical model of the execution time distribution Y of N-ary and binary parallel compositions of stochastic tasks. Our approach is based on approximating the distributions X_i and Y in terms of the first four moments, in conjunction with the use of the GLD as an intermediate vehicle, to derive a closed-form, $O(1)$ complexity expression for $\mathsf{E}[Y^r]$ in terms of $\mathsf{E}[X_i^r]$ and N. Our model is intended to cover the effects of data and task parallelism, and forms an integral part of an analytic, ultra-low complexity approach to parallel program performance prediction that already covers sequential and conditional compositions.

We have also studied to what extent the moment approximation and the GLD approximation has affected the accuracy of our model. Measurements using a large variety of distributions, both synthetically generated as well as empirically obtained from real programs, indicate that the relative error for N-ary composition is in the 1% range for iid workloads, merely increasing slowly with N, while for different workloads the error for binary composition is insensitive to the variation of parameter values. Our results also confirm the fact that the model only degrades when applied to distributions that either exhibit large correlation or that are no longer unimodal. Considering its ultra-low solution complexity, our approach provides an attractive cost-performance trade-off in analytical performance modeling of data-dependent parallel programs.

While the ε_r plots provide ample information on the accuracy of the individual moments, we also investigate the quality of our approach in terms of how well Eq. (5.18) actually predicts the pdf of the actual distribution. To this end, we perform the χ^2 test to fit pdf(Y_p) to pdf(Y_m). Table 5.9 lists the χ^2 values for NAS-EP, PSRS, and WATOR for which the χ^2 value is under or close to the threshold for a 5% significance level. The ratio between the χ^2 values and the threshold is also included for interpretation convenience. As our method typically is used recursively in program performance prediction, we have also included the ε_r values. From the table it can be seen that, besides the low ε_r values, for a quite number of distributions and parameter values, our approximation even passes the χ^2 test, which implies that in those circumstances the distribution Y_p is virtually identical to Y_m.

Table 5.9: Scope of distributions for which the Eq. (5.18) distribution error is virtually negligible

Experiment	ε_1[%]	ε_2[%]	ε_3[%]	ε_4[%]	χ^2/Threshold	Ratio
NAS-EP ($N = 2$)	$1.2\ 10^{-4}$	$2.4\ 10^{-4}$	$3.8\ 10^{-4}$	$5.0\ 10^{-4}$	50.0 / 67.5	0.7
NAS-EP ($N = 4$)	$5.9\ 10^{-4}$	$1.2\ 10^{-3}$	$1.8\ 10^{-3}$	$2.4\ 10^{-3}$	80.7 / 90.5	0.9
NAS-EP ($N = 8$)	$6.4\ 10^{-4}$	$1.3\ 10^{-3}$	$1.9\ 10^{-3}$	$2.5\ 10^{-3}$	55.1 / 55.8	1.0
NAS-EP ($N = 16$)	$3.1\ 10^{-3}$	$6.1\ 10^{-3}$	$9.2\ 10^{-3}$	$1.2\ 10^{-2}$	170.1 / 55.8	3.0
PSRS ($N = 2$, `sort`)	$5.7\ 10^{-2}$	$1.6\ 10^{-1}$	$1.8\ 10^{-1}$	$2.4\ 10^{-1}$	22.7 / 35.2	0.6
PSRS ($N = 4$, `sort`)	$6.4\ 10^{-2}$	$1.3\ 10^{-1}$	$1.9\ 10^{-1}$	$2.5\ 10^{-1}$	33.0 / 43.8	0.8
PSRS ($N = 8$, `sort`)	$5.9\ 10^{-2}$	$1.2\ 10^{-1}$	$1.8\ 10^{-1}$	$2.5\ 10^{-1}$	27.7 / 37.7	0.7
PSRS ($N = 16$, `sort`)	$1.7\ 10^{-1}$	$3.5\ 10^{-1}$	$5.2\ 10^{-1}$	$7.0\ 10^{-1}$	40.9 / 28.9	1.4
PSRS ($N = 32$, `sort`)	$2.4\ 10^{-1}$	$4.7\ 10^{-1}$	$7.1\ 10^{-1}$	$9.6\ 10^{-1}$	32.3 / 25.0	1.3
WATOR ($N = 2, t = 0$)	$3.8\ 10^{-3}$	$7.3\ 10^{-3}$	$1.0\ 10^{-2}$	$1.2\ 10^{-2}$	88.1 90.5	1.0
WATOR ($N = 2, t = 4$)	$3.8\ 10^{-1}$	$7.4\ 10^{-1}$	1.1	1.4	183.2 / 79.1	2.3
WATOR ($N = 4, t = 0$)	$1.6\ 10^{-2}$	$3.2\ 10^{-2}$	$4.7\ 10^{-2}$	$6.0\ 10^{-2}$	75.7 / 79.1	1.0
WATOR ($N = 4, t = 4$)	$2.6\ 10^{-1}$	$5.1\ 10^{-1}$	$7.7\ 10^{-1}$	1.0	132.8 / 55.8	2.4
WATOR ($N = 8, t = 0$)	$1.1\ 10^{-2}$	$2.2\ 10^{-2}$	$3.3\ 10^{-2}$	$4.5\ 10^{-2}$	75.3 / 79.1	1.0
WATOR ($N = 8, t = 4$)	$2.5\ 10^{-1}$	$5.0\ 10^{-1}$	$7.4\ 10^{-1}$	$9.6\ 10^{-1}$	160.8 / 79.1	2.0
WATOR ($N = 16, t = 0$)	$5.7\ 10^{-2}$	$1.1\ 10^{-1}$	$1.7\ 10^{-1}$	$2.3\ 10^{-1}$	93.9 / 101.9	0.9
WATOR ($N = 16, t = 4$)	$2.1\ 10^{-1}$	$4.2\ 10^{-1}$	$6.2\ 10^{-1}$	$8.0\ 10^{-1}$	134.1 / 90.5	1.5
WATOR ($N = 32, t = 0$)	$9.1\ 10^{-2}$	$1.8\ 10^{-1}$	$2.7\ 10^{-1}$	$3.6\ 10^{-1}$	110.5 / 79.1	1.4
WATOR ($N = 32, t = 4$)	$3.6\ 10^{-2}$	$4.8\ 10^{-2}$	$3.5\ 10^{-2}$	$5.9\ 10^{-3}$	141.7 / 67.5	2.1
WATOR ($N = 64, t = 0$)	$1.3\ 10^{-2}$	$2.7\ 10^{-2}$	$4.2\ 10^{-2}$	$5.8\ 10^{-2}$	78.4 / 43.8	1.8
WATOR ($N = 64, t = 4$)	$2.0\ 10^{-2}$	$6.0\ 10^{-2}$	$1.2\ 10^{-1}$	$2.1\ 10^{-1}$	153.8 / 43.8	3.5
WATOR ($N = 128, t = 0$)	$1.7\ 10^{-1}$	$3.4\ 10^{-1}$	$5.2\ 10^{-1}$	$6.9\ 10^{-1}$	224.1 / 67.5	3.3

Chapter 6

Tool Implementation

In this chapter, we present an implementation of our methodology in terms of a process-oriented modeling language and compiler. The language is called PAMELA$^+$ of which a prototype version, PAMELA (PerformAnce ModEling LAnguage [28]), based on the use of deterministic variables, has been described in [29]. We choose to extend PAMELA to implement our methodology because PAMELA also features symbolic analysis which is reminiscent to ordinary mathematics in the sense that deriving formulae for the execution time is straightforward. In contrast to PAMELA, however, PAMELA$^+$ fully supports the use of *stochastic* variables. Furthermore, the methodology is supported by a public-domain research *compiler* that automatically translates a PAMELA process model into an analytic model that predicts the first four moments of the execution time at minimum cost. In this chapter we will focus on the symbolic evaluation and transformation capabilities of the compiler rather than on the low-level implementation details.

The chapter is organized as follows. In Section 6.1 we summarize the main syntax and semantics of PAMELA. In Section 6.2 we present the stochastic extensions in PAMELA$^+$ and the associated compiler implementation issues. In Section 6.3 we describe the accuracy of PAMELA$^+$ with respect to condition synchronization and mutual exclusion. In Section 6.4 we present an application case study.

6.1 PAMELA

6.1.1 Formalism

PAMELA is a process algebraic language that allows a parallel program-machine combination to be modeled in terms of a sequential, conditional, and parallel composition of processes which are synchronized in terms of condition synchronization and resource contention. A PAMELA program or process (whose root process is usually denoted by L) is written as a set of algebraic equations that constitute the entire model. Hereafter, we will often refer to a PAMELA program as a PAMELA model.

In PAMELA, work is described by the **use** process construct, like in **use**(s, τ), in which the invoking process exclusively acquires service from a server s for τ units time (workload, excluding possible queuing delay). A resource s has a multiplicity, denoted by $|s|$, which may be larger than 1. In simulation mode, the service time τ may be deterministic or

stochastic. In symbolic analysis mode, only deterministic values are considered. Like in queuing networks, it is convenient to define an infinite server ρ such that $|\rho| = \infty$. Instead of $\mathbf{use}(s, \tau)$, we can simply write $\mathbf{delay}(\tau)$.

PAMELA provides process composition operators for

- Sequential composition, which includes the binary operator $';'$ as in $L_1 \,; L_2$, which defines a sequential composition of submodels L_1 and L_2, and the replication operator **seq** as in **seq** $(i = a, b)$ L_i, which equals $L_a \,; \ldots \,; L_b$.
- Parallel composition, which includes the binary operator $'\parallel'$ as in $L_1 \parallel L_2$, which defines a parallel composition of submodels L_1 and L_2 including barrier synchronization at the exit, and the replication parallel operator **par** as in **par** $(i = a, b)$ L_i, which equals $L_a \parallel \ldots \parallel L_b$.
- Conditional composition, as in **if** (c) L, where c is a stochastic condition. For programming convenience, an **else** construct is provided.

The implicit condition synchronization of the sequential and parallel composition constructs allow for the expression of models that are similar to SP-DAGs. We conclude this section by an example that forms a typical demonstration of the modeling approach.

Consider a machine repair model (MRM)) [56] in which P clients either spend τ_l on local processing or request service from a single First Come First Served (FCFS) server s with service time τ_s for a total cycle count of N iterations (unlike the steady-state analysis typical for queuing systems or Petri nets, in our approach we require models to terminate). The PAMELA model of the MRM is given by

$$
\begin{array}{l}
L = \mathbf{par}\ (p = 1, P) \\
\qquad\quad \mathbf{seq}\ (i = 1, N)\ \{ \\
\qquad\qquad \mathbf{delay}(\tau_l)\,; \\
\qquad\qquad \mathbf{use}(s, \tau_s) \\
\qquad\quad \}
\end{array}
$$

For reading convenience we have given the above process-algebraic expression L in program format including the usual indentation.

The example illustrates the top-down, *material-oriented* modeling approach [53] taken in PAMELA, in which the server is modeled by a *passive* resource. In a *machine-oriented* approach (the dual modeling paradigm), the server would have been modeled by a separate *process* that synchronizes through explicit message passing such as in Communicating Sequential Processes (CSP) [41]. Unlike our approach however, the latter paradigm is not amenable to automatic analysis process where a PAMELA model is *automatically compiled* to analytic execution time expressions. This is due to the inherent difference in modeling condition synchronization and mutual exclusion. In message-passing paradigms, both forms of synchronizations are implicitly expressed through the *same* constructs, i.e., synchronous communication, combined with a non-deterministic choice operator (selective communications). This makes it impossible to symbolically and automatically express the separate timing effects of both synchronization types, which is the basis of PAMELA's approach. In the PAMELA top-down modeling approach, problem parallelism (including

condition synchronization) is modeled explicitly in terms of **par** (or $\|$) operators, explicitly constrained by mutual exclusion (**use** as a result of scheduling when the processes need to share resources such as, e.g., software locks, file servers, processors, communication links, memories, and/or I/O disk handlers).

6.1.2 Symbolic Analysis

In this section, we summarize the symbolic analysis technique for two synchronization types, i.e., condition synchronization and mutual exclusion (for details we refer to [29]). It is necessary to describe the symbolic analysis technique here since the next section will show how we modified this scheme in order to support our moment approach. The symbolic analysis approach is based on a lower bound approximation of contention delays, integrated within a condition synchronization delay analysis similar to critical path analysis of SP-DAGs. Let L denote a PAMELA model and let $T(L)$ denote its execution time (i.e., the result of simulating L). To keep the description simple we present the analysis for unconditional models. Conditional composition is incorporated in the compilation scheme by simply transferring the condition to the time domain according to the symbolic transformation

$$T(\textbf{if } (c)\ L) = \langle c \rangle\ T(L), \tag{6.1}$$

where $\langle c \rangle$ returns 1 if c is true and 0 otherwise (see also Eq. (2.1)).

Let L denote a PAMELA model comprising some sequential and parallel composition of **use** or **delay** tasks. One estimate of the execution time of L is given by computing the effects of condition synchronization, based on a critical path analysis in which we ignore the effect of contention (i.e., each **use** statement is interpreted as if it were a **delay** statement). Let $\varphi(L)$ denote the critical path estimate. In terms of the binary PAMELA operator ';' and '$\|$', the following symbolic transformation applies

$$\varphi(L) = \begin{cases} \varphi(L_1) + \ldots + \varphi(L_N), & L = L_1\,;\,\ldots\,;L_N, \\ \varphi(L_1) \max\ \ldots \max\ \varphi(L_N), & L = L_1 \,\|\, \ldots \,\|\, L_N, \\ \tau, & L = \textbf{delay}(\tau) \text{ or } L = \textbf{use}(r,\tau). \end{cases} \tag{6.2}$$

As mutual exclusion is disregarded, it holds that $\varphi(L) \leq T(L)$.

Another estimate of $T(L)$ is given by computing the effect of mutual exclusion in which we use the fact that, regardless of the preemptive or non-preemptive scheduling order in which processes access resources, the total time spent per resource establishes a lower bound on T. Let $\underline{\delta}(L) = (\delta_1, \ldots, \delta_M)$ denote the total service demand vector of L, where M is the total number of resources involved and δ_m denotes the service demand on resource r_m. We will write $\delta_m(L)$ to denote the m-th element of $\underline{\delta}(L)$. Clearly,

$$\underline{\delta}(L) = \begin{cases} \underline{\delta}(L_1) + \ldots + \underline{\delta}(L_N), & L = L_1\,;\,\ldots\,;L_N \text{ or } L = L_1 \,\|\, \ldots \,\|\, L_N, \\ \tau \underline{e}^m, & L = \textbf{use}(r_m,\tau). \end{cases} \tag{6.3}$$

where $\underline{e}^m = (0, \ldots, 0, 1, 0, \ldots, 0)$ is the M-dimensional unit vector in the m direction, and addition and multiplication are defined element-wise. Let ω denote the lower bound on the execution time of L due to mutual exclusion. Then

$$\omega(L) = \max_{m=1\ldots M} \frac{\delta_m(L)}{\mid r_m \mid}. \tag{6.4}$$

As condition synchronization is disregarded, this results in $\omega(L) \leq T(L)$.

If the lower bound due to contention (ω) is combined with the result of critical path analysis (φ), it follows that the lower bound on T_L is predicted by

$$T^l = \max(\varphi(L), \omega(L)), \tag{6.5}$$

where Eq. (6.5) applies to simple parallel sections (involving a single **par**). For more complex models, the following recursive generalization provides a much sharper bound:

$$T^l(L) = \begin{cases} T^l(L_1) + \ldots + T^l(L_N), & L = L_1 \,;\, \ldots \,;\, L_N, \\ \varphi(L_1) \max \ \ldots \max \ \varphi(L_N) \max \ \omega(L), & L = L_1 \parallel \ldots \parallel L_N, \\ \max(\varphi(L), \omega(L)), & \text{otherwise.} \end{cases} \tag{6.6}$$

Eq. (6.6) shows that the two synchronization types are mapped to the mathematical operators '+' and 'max'. The above process has a linear solution time complexity.

Recall the MRM example. By Eqs. (6.2) and (6.4), it follows that

$$\varphi = \max_{p=1\ldots P} \sum_{i=1}^{N} (\tau_l + \tau_s) = N(\tau_l + \tau_s) \tag{6.7}$$

and

$$\omega = \sum_{p=1} P \sum_{i=1}^{N} \tau_s = PN\tau_s. \tag{6.8}$$

Hence, by Eq. (6.5) or Eq. (6.6), it follows that

$$T^l = N \max(P\tau_s, \tau_l + \tau_s). \tag{6.9}$$

Unlike traditional static DAG analysis, T^l accounts for the additional queuing delay when s is saturated. The above analysis yields the same result as asymptotic bound analysis in queuing theory [116]. Let R denote the response time and let $Z = \tau_l$ denote the local time. Then the mean cycle time $R + Z$ equals φ/N for $P \ll P^*$ and ω/N for $P \gg P^*$, where the saturation point $P^* = (\tau_s + \tau_l)/\tau_s$ denotes the crossover between the asymptotes.

6.1.3 Implementation

The Pamela language and associated symbolic analysis is implemented in terms of a source-to-source compiler that translates a Pamela model into a symbolic execution time prediction. The internal intermediate representation of the compiler is implemented using Tm [107].

A Pamela model is a list of equations of the form

```
type [parameter] identifier [paramlist] [= expression]
```

which defines an identifier by an algebraic expression, possibly with argument parameters, of type `numeric`, `process`, or `resource`. The identifier can be annotated as parameter using the optional `parameter` modifier, which is explained later on. In this case, the expression can be omitted, effectively yielding an identifier declaration, rather than an equation. Returning to the MRM example, the Pamela model is coded as follows:

```
numeric P = 1000                          % # clients
numeric N = 1000000                       % # iterations
numeric tl = 10
numeric ts = 0.1
resource s = fcfs(0,1)
process main =   par (p = 1, P)           % fork P clients
                     seq (i = 1, N) {     % each loops N times
                         delay(tl) ;      % something local
                         use(s,ts)        % obtain service
                     }
```

where `fcfs` is a predefined FCFS resource array in which the first and second argument denote the index and its multiplicity, respectively. For each identifier used in the expressions there must be a matching definition/parameter declaration.

Apart from the process operators mentioned above, PAMELA includes the usual unary and binary numeric operators such as `+`, `*`, `/`, `mod`, `div`, `==`, `<`, `max`, etc., as well as the reduction operators `sum (<index> = <lb>, <ub>)` and `max (<index> = <lb>, <ub>)`. Conditional numeric expressions are described using `if-then` just like in conditional process expressions. Furthermore, as parts of the analysis result are expressed in terms of vectors, the `numeric` abstract data type includes vectors as well as scalars, implying that all numeric operators are overloaded. A vector is denoted as `[<scalar>, ..., <scalar>]`. Hence, the expression `[1,2,3] * 4` is legal and, incidentally, will be compiled to `[4,8,12]`. In order to generate unbounded symbolic vectors, PAMELA features the `unitvec` operator, which returns a unit vector in the dimension (base 0) given by its argument. For instance, the expression τ `* unitvec(3)` will be compiled to `[0,0,0,`τ`]`.

Compilation

The compiler is typically used in terms of the following pipeline

```
pampp | pamparse | pameval | pam2time | pameval | pamprint
```

Apart from the first pipe, which carries PAMELA source code, all intermediate pipes carry the Tm representation. The compiler is implemented using the C programming language in terms of the following modules (comprising more than 8,000 lines of source code):

- preprocessor `pampp`.
  Supports file inclusion (`include` statement) and removes comments (% starts comment until end of line).

- parser `pamparse`.
  Parses PAMELA sources, generating an intermediate Tm representation. The parser is implemented using Flex [76] and Bison [20].

- evaluator `pameval`.
  Implements expression evaluation, which involves substituting expressions where possible as well as numerically evaluating expressions where possible. The evaluator is used at least at two stages of the performance modeling process (see above

script): (1) before transformation to time, in order to perform all necessary resource expression substitutions, (2) to further optimize the expressions resulting from the transformation process. The resulting model is the model returned to the user. Typically, however, the user will again call the evaluator after assigning all or some of the remaining symbols (parameters) to a numeric value.

- analyzer `pam2time`.
  Transforms process and resource equations into numerical equations according to the calculus described in Section 6.1.2.

- unparser `pamprint`.
  Prints PAMELA equations from Tm representation back into PAMELA source grammar. This is the readable form in which the results are fed back to the user.

The above example model is compiled as follows. After the first `pameval` pass, the following model results (the intermediate `.tm` output shown has been unparsed by `pamprint`):

```
numeric P = 1000
numeric N = 1000000
numeric tl = 10
numeric ts = 0.1
resource s = fcfs(0,1)
process main =   par (p = 1, 1000)
                     seq (i = 1, 1000000) {
                         delay(10) ;
                         use(fcfs(0,1),0.1)
                     }
```

In process `main`, symbols `P` and `N` have been substituted by their right-hand sides (rhs). If `P` and `N` would involve evaluable expressions, `pameval` would evaluate these expressions immediately. Resource `s` has been substituted by the `fcfs` definition.

At this point, the process equation is in a form suitable for transformation into the corresponding time equations using `pam2time`. For each PAMELA process, four performance models are generated, i.e., T^l, φ, δ and ω, all of which are explained in Section 6.1.2. Thus for some `process` identifier `L`, the compiler will generate the identifier `T_L`, `phi_L`, `delta_L` and `omega_L`. After the first `pameval` pass, the following model results (again, the `.tm` output shown has been unparsed by `pamprint`)

```
numeric P = 1000
numeric N = 1000000
numeric tl = 10
numeric ts = 0.1
numeric T_main = max(max (p = 1, 1000) {
                     sum (i = 1, 1000000) {
                         (10 + (0.1 / 1))
                     }
                 },max(sum (p = 1, 1000) {
                           sum (i = 1, 1000000) {
                               ((0.1 / 1) * unitvec(0))
```

```
                            }
                  }))
numeric phi_main = max (p = 1, 1000) {
                       sum (i = 1, 1000000) {
                           (10 + (0.1 / 1))
                       }
                   }
numeric delta_main = sum (p = 1, 1000) {
                         sum (i = 1, 1000000) {
                             ((0.1 / 1) * unitvec(0))
                         }
                     }
numeric omega_main = max (sum (p = 1, 1000) {
                               sum (i = 1, 1000000) {
                                  ((0.1 / 1) * unitvec(0))
                               }
                           })
```

Finally, the second `pameval` pass yields the following time domain model:

```
numeric P = 1000
numeric N = 1000000
numeric tl = 10
numeric ts = 0.1
numeric T_main = 100000000
numeric phi_main = 10100000
numeric delta_main = 100000000 * unitvec(0)
numeric omega_main = 100000000
```

Thus, as in ordinary mathematics, the semantics of a PAMELA model is based on an algebraic model. Although for readability a model may be coded in terms of many equations, internally each expression is evaluated by recursively substituting every global variable by its corresponding rhs expression.

Parameterization

By virtue of the symbolic nature of the analysis process, PAMELA models are typically parameterized, thus parameters are preserved in the resulting time domain model. In order to enable parameterization, PAMELA supports the `parameter` modifier, which blocks the substitution process from attempting to substitute a rhs expression for the particular variable that is intended to become a parameter. Consider the MRM model presented earlier. As all variables are bound to numeric values, the compiler will already evaluate the model for the given values of `P`, `N`, `tl`, and `ts`. In order to investigate the effect of `P`, `N`, `tl`, and `tc`, however, we want to redefine these variables according to

```
numeric parameter P
numeric parameter N
numeric parameter tl
numeric parameter ts
```

Symbolic compilation now yields the intermediate result (`T_main` shown only)

```
numeric T_main = max(max (p = 1, P) {
                          sum (i = 1, N) {
                              (tl + (ts / 1))
                          }
                      }, max(sum (p = 1, P) {
                                 sum (i = 1, N) {
                                     ((ts / 1) * unitvec(0))
                                 }
                             })
```

Although parameterized, this model is automatically and symbolically simplified as a result of simple transformation rules, such as

```
sum (i = a, b) scalar = (b-a+1) * scalar
```

yielding the following symbolic performance model:

```
numeric T_main = max(N * (tl + ts), P * N * (ts))
```

which is in accordance with Eq. (6.9). This model can now be evaluated for different values of `P`, `N`, `tl`, and `ts`, possibly using mathematical tools other than the PAMELA compiler. In PAMELA, further evaluation is achieved as described earlier, by recompiling the above result after removing one or both `parameter` modifiers while providing a numeric rhs expression. Note that the evaluation cost may decrease by orders of magnitude due to regularity of model.

6.2 PAMELA$^+$

In this section we present an extension of the compiler, denoted as PAMELA$^+$, which supports our statistical moments analysis technique. The extension involves modification of `pamparse`, `pameval` and `pamprint` (comprising more than 500 additional lines of source code). PAMELA$^+$ is available in the public domain [74].

In `pamparse`, we define stochastic workload using the `moments` operator, which returns a vector of the type `numeric` consisting of exactly 4 scalars. For instance, `moments(m1,m2,m3,m4)` denotes a stochastic workload with mean `m1`, variance `m2`, skewness `m3`, and kurtosis `m4`. Consequently, in PAMELA$^+$, the eventual results are also expressed in terms of `moments`.

In order to evaluate operations involving `moments` variables, we redefine the operators given in Table 6.1 by modifying `pameval`. When one or more operands have non-zero variance, the corresponding equations in the table will be applied, otherwise the operands are treated as (deterministic) scalars. Therefore, the operators must test the non-zero variance of the operands. Additionally, when we deal with the reduction operators `sum (<index> = <lb>, <ub>)` and `max (<index> = <lb>, <ub>)`, it is necessary to check the independence between the expression and the corresponding `index`. If a dependence occurs, we apply the binary equations Eqs. (3.45) and (5.35) rather than the N-ary equations Eqs. (3.45) and (5.17), respectively. For the conditional operator, we express the probability of `cond` in terms of moments, as shown by the following example:

Table 6.1: Modified operators

Operator	Equation
binary `+`	(3.36)
`sum (<index> = <lb>, <ub>)`	(3.45)
`if (cond) ... else ...`	(3.53)
`switch`	(3.55)
binary `max`	(5.35)
`max (<index> = <lb>, <ub>)`	(5.18)

Program:

```
for (i = 1; i <= N; i++)
    if (x[i] != 0)
        x[i] = x[i] * alpha;
```

PAMELA$^+$ model:

```
seq (i = 1, N)
    if (moments(5e-3, 2e+0, 4e+0, 2e+1))
        delay(moments(1,1,2,9))
```

The program (left) shows a conditional statement in a loop with bound N. In the example (right) we show the corresponding PAMELA$^+$ model for the truth probability of `x[i] != 0` and the execution time of `x[i] = x[i] * alpha;` given by `moments(5e-3, 2e+0, 4e+0,2e+1)` and `moments(1,1,2,9)`, respectively. To evaluate the total moments of the PAMELA$^+$ model, we use Eqs. (3.53) and (3.45) respectively. As shown in the PAMELA$^+$ model above, the user can print a readable form of the model involving the `moments` operator. To make this feature possible we have modified `pamprint` such that the `moments` operator can be unparsed from the intermediate Tm representation.

In the following, we present the PAMELA$^+$ model for the MRM example:

```
numeric exponential(mu) = moments(mu,mu*mu,2,9)  % appr exp distr
numeric parameter P                              % # clients
numeric parameter N                              % # iterations
numeric tl = exponential(10)                     % exp (mu = 10)
numeric ts = exponential(0.1)                    % exp (mu = 0.1)
resource s = fcfs(0,1)
process main =   par (p = 1, P)                  % fork P clients
                     seq (i = 1, N) {            % each loops N times
                         delay(tl) ;             % something local
                         use(s,ts)               % obtain service
                     }
```

Symbolic compilation now yields the intermediate result (`T_main` shown only)

```
numeric T_main = max(max (p = 1, P) {
                          sum (i = 1, N) {
                              (moments(10,100,2,9) +
                               moments(0.1,0.01,2,9))
                          }
                     },max(sum (p = 1, P) {
                                sum (i = 1, N) {
```

```
                                    moments(0.1,0.01,2,9) * unitvec(0)
                                }
                        })
```

As P and N have not yet been bound to numeric values, the symbolic simplification engine within the PAMELA$^+$ compiler yields the following cost model, which is the final compilation result:

```
numeric parameter P
numeric parameter N
numeric T_main = max(max (p = 1, P) {
                          N * moments(10.1,100.01,2,9)
                        },(P * N * (moments(0.1,0.01,2,9))))
```

Note that in contrast to the deterministic simplification in PAMELA, in PAMELA$^+$ the max reduction has not yet been symbolically simplified because of the order statistical evaluation that is involved (Eq. (5.18)). The above result can be subsequently evaluated for different values of P and N. Using the PAMELA$^+$ compiler, further evaluation is simply achieved by recompiling the above model after removing parameter modifiers and providing a numeric rhs expression. For example, the following modification

```
numeric P = 1000
numeric N = 1000000
```

causes the model to be compiled (i.e., evaluated) to

```
numeric T_main = moments(1.01+07,4.11+03,4.69-01,3.23+00)
```

During this evaluation process, Eqs. (3.36), (3.45), (5.18) and (5.35) are used.

As described in Section 6.1.1, PAMELA supports two synchronization types, i.e., condition synchronization and mutual exclusion. In the symbolic analysis, both synchronizations are mapped to the mathematical operator '+' and 'max'. Note that we have implemented our moment approach in PAMELA$^+$ without discriminating whether the mathematical operator '+' and 'max' come from a different synchronization type, while our moment approach applies only to the analysis of condition synchronization. The consequences in terms of accuracy are discussed in Section 6.3.

To conclude the example, we derive the mean cycle time of the MRM as function of P by dividing T_main by N. The resulting prediction T_{MOM} = T_main/N is shown in Table 6.2. As a comparison to T_{MOM}, we compute T analytically using Mean Value

Table 6.2: MRM mean cycle time

P	1	2	5	10	20	50	100	200	500
T^l	10.10	10.10	10.10	10.10	10.10	10.10	10.10	20	50
T_{MVA}	10.10	10.10	10.11	10.12	10.12	10.19	10.82	20	50
T_{MOM}	10.10	10.11	10.12	10.12	10.12	10.12	10.13	20	50

Analysis (MVA) [85], denoted by T_{MVA} (as tl and ts are approximately exponential).

Since the MRM maps to separable queuing networks [13], MVA may be applied, which yields

$$T_{\text{MVA}} = N(R(P) + \tau_l), \tag{6.10}$$

where the response time $R(n)$ for the server for n clients in the closed system is given by the MVA recursion [57]

$$R(0) = 0, R(n+1) = \left[1 + \frac{PR(n)}{\tau_l + R(n)}\right] \tau_s. \tag{6.11}$$

In the table we also compare the results with the deterministic lower bound analysis T^l in Eq. (6.9) using the mean values of `tl` and `ts`.

Table 6.2 illustrates the effects of the low-cost approximation in PAMELA$^+$ of the effects of mutual exclusion (queuing). While the prediction error ε_1 for $P = 0$ and $P \to \infty$ is zero, near the saturation point ($P = 100$) the error is around 8%. Experiments [30] indicate that for very large systems ($O(1000)$ resources) the worst-case average error is limited to 50% and only occurs in the region near the saturation point (the cross-over region). However, these situations seldom occur: typically, systems are either dominated by condition synchronization or mutual exclusion, in which case the approximation error is in the percent range.

Although in this particular example the introduction of variance in `tl` and `ts` has a negligible effect on the prediction of $\mathsf{E}[T]$, unlike T^l and T_{MVA}, T_{MOM} provides approximations of $\mathsf{Var}[T], \mathsf{Skw}[T]$, and $\mathsf{Kur}[T]$, respectively, from which the distribution of T can be approximated. Furthermore, based on the distribution approximation, we can show that T tends to go to the normal distribution since the value of N is large. For small P the prediction values of T_{MOM} are slightly higher than the deterministic lower bound $T^l = \tau_l + \tau_s$ due to the order statistics effect computed by Eq. (5.18).

The effect of our $O(1)$ symbolic analysis technique on the solution complexity is profound. For instance, on a 1 GHz Pentium III, the symbolic performance model of the MRM merely requires 0.3 seconds CPU time per point (irrespective of N and P), where most of the CPU time is required to evaluate Eqs. (5.17) and (5.35). In contrast, similar to PAMELA, the evaluation of the model before symbolic simplification would take approximately 7,000 seconds CPU time per point, where most of the CPU time is required to `P*N` times evaluate Eq. (3.36). The $O(10^4)$ time reduction provides a compelling case for the use of symbolic cost models in performance prediction. Solution time reductions in excess of 10^8 have been reported in [32].

6.3 Accuracy

As mentioned before, in principle our moment method is targeted to the analysis of condition synchronization. The previous example, however, included mutual exclusion. As our analysis applies to all '+' and 'max' operators, the effects of mutual exclusion are therefore approximately covered. In this case, however, we are confronted with the fact that a correlation occurs between ϕ and ω in Eq. (6.5) since ϕ and ω are obtained from the same operands. For example, for `main = use(s,ts)`, where `s = fcfs(0,1)` and `ts = moments(0.1,0.01,2,9)`, the final prediction yields `T_main = max(ts,ts)`. In

PAMELA$^+$, the prediction overestimates the actual execution time due to the independency assumption in Eq. (5.35). In the example, the overestimation depends on the variance of `ts`. Since we deal with operands having limited variance, the overestimation is also limited.

Another example of overestimated prediction is also shown in Table 6.2 for MRM, where $P = 5$ due to the overestimation of Eq. (6.4). Clearly, when we apply our moment method to the analysis of mutual exclusion, the prediction is no longer guaranteed to be a lower bound. On the other hand, the overestimation by including mutual exclusion in our analysis is quite limited. Note that the correlation issue with φ and ω only applies when their first four moments are equal (i.e., in the cross-over region). This region, however, is the area where the prediction accuracy is already limited due to the lower bound (as explained in Section 6.2). Consequently, the correlation effect does not degrade the essentially approximate technique, while for models including condition synchronization only (or mutual exclusion only), our technique fully applies.

6.4 Modeling Example

In this section we present an example of how PSRS can be modeled using PAMELA$^+$. We model the algorithm for a shared-memory architecture, including the effect of memory communication delays.

In the PAMELA$^+$ model of PSRS, given in Figure 6.1, some numeric variables are declared in the first seven lines of the model, i.e., the array length `N` and the processor number `P`. The moments of `c1`, `c2`, and `c3` are obtained from profiling, where the parameter `c` in `sort` must be profiled separately for each function call. The next lines declare the CPU resources and the shared memory resource. Although in the current model the workload of each parallel process is mapped onto a unique CPU (see the `flop` model), the explicit use of CPU resources also allows the modeler to investigate the effects of multithreading, where multiple tasks may be mapped on the same CPU. Note that the `shmem` multiplicity is currently set to a value such that no contention will ever occur as our moments method focuses on condition synchronization. The evaluation of the model starts from the `main` process, which consists of 3 parallel and 2 sequential sections. In order to simplify the presentation, we have chosen Straight Selection Sort instead of Quicksort (which requires a much larger sequential model). The `main` process calls the other 3 processes, i.e., the program (sub)model `sort` (Straight Selection Sort), and the machine models `flop` and `move`, which model the floating-point workload and the shared memory load/store workload, respectively. The latter 2 processes represent workloads with exponential time delay `tf` and `tm`. Note that the `move` interface allows the possibility of modeling, e.g., caching or memory contention without requiring further model modification.

In our experiments we use random input vectors of size `N` $= 81,920$. Instead of comparing our predictions with the run times of the actually running parallel program (which would require a much more detailed PAMELA$^+$ model) we compare our prediction to a multithreaded version of PSRS that has been instrumented with counters to measure the actual execution time. The predicted execution time for `P` ≥ 2 is obtained from profiling and running the program in Figure 6.1. while for `P` $= 1$ we choose Straight

selection sort as the sequential program, instead of simply running PSRS for `P` = 1 (i.e., `process sort` in Figure 6.2).

To observe the effect of memory communication on the speedup we consider both `tm` = 0 and `tm = moments(10,100,2,9)`. In Figure 6.2, the measured and predicted speedup are shown by solid and dashed lines, respectively, for `P` $\leq$ 128. The speedup increases linearly with the number of processors for `P` $\leq$ 64 and decreases for large `P` since the working of PSRS becomes inefficient for small array sizes. For `tm = moments(10,100,2,9)`, the speedup degrades such that for `P` = 2 the speedup is less than 1. This is caused by the fact that for $P \geq 2$ the parallel version generates much more communication in `disjoin` and `sort2` than for $P = 1$, while the degree of parallelism is still low for `P` = 2. For both values of `tm` the speedup prediction is approximately accurate while the error is mainly caused by the inaccuracy in profiling `c`. Note that the predicted execution times are very sensitive to the value of `c`, since `c` is located in a loop nest with high invocation frequency.

The corresponding ε_1 of Figure 6.2 is shown in Figure 6.3. The highest error is $\varepsilon_1 = 18\%$ at `P` = 16 for `tm` = 0 and $\varepsilon_1 = 15\%$ at `P` = 2. ε_1 decreases as `P` increases since our measurements show that the correlation in `sort` is high for large array size. The error is quite acceptable and the evaluation of the compiled version of the above model only requires 0.6 seconds CPU time on a 1 GHz pentium III.

6.5 Summary

In this chapter we have presented a tool implementation of our statistical moment approach. The tool has been coined the PAMELA$^+$ compiler. The PAMELA$^+$ compiler provides modeling support for predicting program execution time by automatically generating analytic performance models, and automatically producing the first four moments. Our experimental results show that for models with condition synchronization-only, the prediction error is in the few percent range, while the evaluation of the models only takes a few seconds. For models including mutual exclusion, our technique delivers results comparable to the approximate, lower-bound prediction of the former deterministic tool version (PAMELA).

```
 numeric parameter N                              % array length
 numeric parameter P                              % # processors
 numeric tf = moments(1,1,2,9)                    % computation time
 numeric tm = moments(10,100,2,9)                 % communication time
 numeric c1 = moments(5e-3, 2e+0, 4e+0, 2e+1)     % profiled for # runs
 numeric c2 = moments(1e-2, 3e-2, 9e+0, 7e+1)     % profiled for # runs
 numeric c3 = moments(4e-3, 5e+0, 2e+0, 6e+0)     % profiled for # runs
 resource cpu(p) = fcfs(p,1)                      % the P CPUs
 resource shmem = fcfs(P,P)                       % shared memory

 process main = par (p=0, P-1) {                  % sort1:
                  sort(p*N/P,(p+1)*N/P-1,p,c1) ;  % sort subarray
                  seq (i=0, P-1)                  % collect samples
                    move(p)
                } ;
                sort(0,P*P-1,p,c2) ;              % sort samples
                seq (i=0, P-2) {                  % choose pivots
                  move(p)
                } ;
                par (p=0, P-1) {                  % disjoin:
                  seq (i=0, P-1) {
                    seq (j=0, (N-N/P)/(P*P)-1) {
                      flop(p) ;
                      move(p)
                    }
                  }
                } ;
                par (p=0, P-1) {                  % sort2:
                  seq (i=0, P-1) {                % collect subarray
                    seq (j=0, (N-N/P)/(P*P)-1)
                      move(p)
                  } ;
                  sort(p*N/P,(p+1)*N/P-1,p,c3)    % sort subarray
}

 process sort(lb,ub,p,c) = seq (i=lb+1, ub) {     % sort
                             seq (j=lb, i-1) {
                               if (c)
                                 flop(p)
                             } ;
                             move(p)              % swap
                           }

 process flop(p) = use(cpu(p),tf)                 % computation
 process move(p) = use(shmem,tm)                  % memory communication
```

Figure 6.1: PAMELA$^+$ model of PSRS

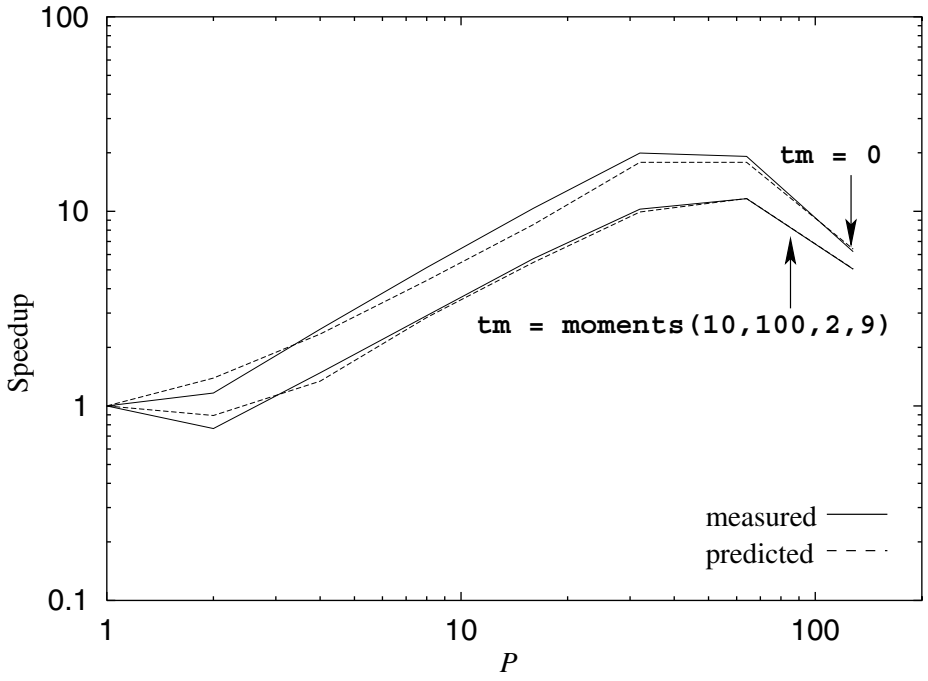

Figure 6.2: Speedup for PSRS

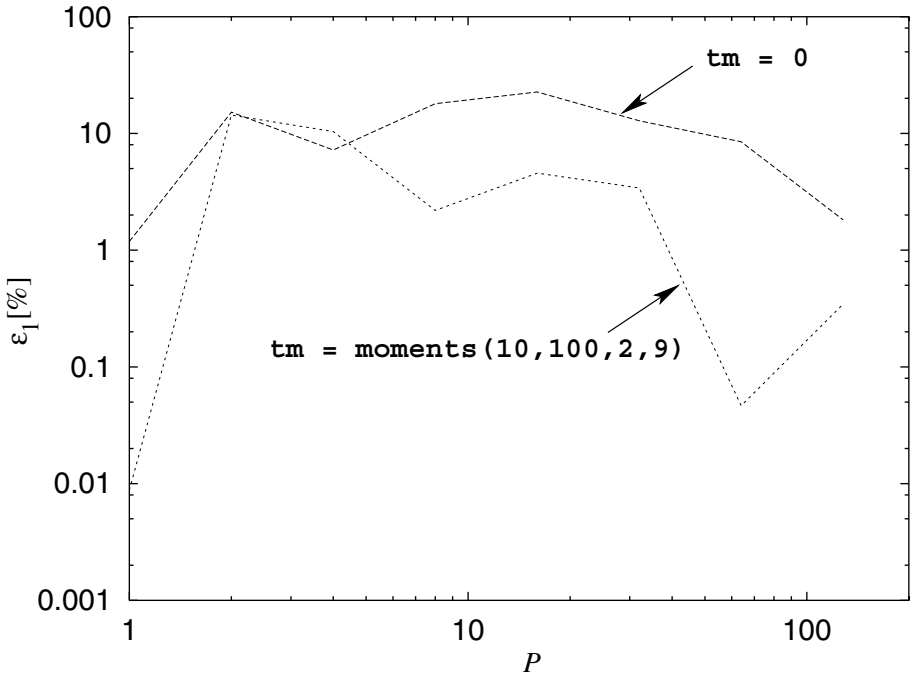

Figure 6.3: ε_1 [%] for PSRS

Chapter 7

Conclusion

In this thesis we have presented a probabilistic approach to the symbolic performance modeling of parallel computer systems. In order to meet the challenges, i.e., a symbolic, closed-form solution that accounts for stochastic program behavior, having low solution complexity, and high prediction accuracy, we have introduced a new analytical approach based on statistical moment distributions. Our approach is aimed at analyzing the stochastic behavior of data-dependent parallel programs across the entire system parameter space and for a wide variety of input data. Our approach provides a general model that effectively captures the distribution of synthetic as well as real-life workloads and applies to sequential, conditional, and parallel compositions. Furthermore, our approach is aimed at providing a balanced trade-off between reasonable prediction accuracy and low solution complexity.

7.1 Contributions

We summarize our major contributions as follows.

- Modeling
  We have introduced a workload modeling technique based on the use of statistical moments as workload representation. For parallel composition, we represent stochastic workload using GLD as intermediate vehicle such that our moment approach can be applied. Apart from modeling sequential and parallel composition based on SP DAGs, we have presented how branches can be modeled statistically. In particular, we have introduced the ARP approach which accounts for some forms of correlation between individual branch invocations.

- Analysis
  We have presented the symbolic analysis of sequential, conditional, and parallel compositions which can be integrated successfully within our compositional approach. As a result, we can predict the execution time distribution for arbitrary compositions. Under workload independence and unimodality assumptions we have derived exact formulae for the execution time distribution of sequential and conditional composition, and derived an approximation formulae for the first four moments of the execution time distribution of parallel composition. In the analysis of parallel

composition, we have succeeded to symbolically solve the order statistics integration problem.

Some restrictions under which our moment approach applies are summarized as follows. We have represented a parallel program in terms of an SP-DAG of which the modeling power is limited to modeling condition synchronization. Modeling stochastic workloads using statistical moments is restricted to unimodal workloads. In particular, statistical moments cannot represent the Laplace distribution [22]. In our moment analysis, we assume that all parameters are independent random variables. Our moment analysis does not apply to non-tail recursive functions.

Apart from the mathematical framework, our contributions also cover the following associated performance modeling aspects:

- Related work
  We have presented a survey on probabilistic approaches to symbolic performance modeling. We have classified the approaches based on two important aspects, i.e., control-flow modeling and stochastic workload representation. We have shown that the approaches do not cover arbitrary composition and/or arbitrary workload distribution. The approaches have either high solution cost, or limited application to a small range of distributions. Many of those suffer from insufficient accuracy.

- Measurement
  To validate our moment approach, we have measured the execution frequency of control flows as well as the total program execution time. We have shown how parallel programs are instrumented based on counter-based profiling. Using the profiling technique we can obtain the moment values in a straightforward manner while we keep our analysis method machine-independent. In the profiling, we also have dealt with how to generate and select representative input data.

- Case studies
  We have applied our moment approach using synthetic workloads, i.e., uniform, exponential and normal distributions, as well as real-life workloads from many test applications, including well-known benchmarks which exhibit various forms of input data related non-determinism: Vector Scaling (a branch in a loop which exhibits Bernoulli behavior), Straight Selection Sort (a branch which behaves other than Bernoulli), Memory Hierarchy (next to branches and loop bounds, machine level non-determinism), NAS-EP (iid parallel composition having normal distributed workload), SSSP (lightly correlated parallel tasks), PSRS (three fork/join parallel tasks with increasing correlation), WATOR (data parallel with extreme workload imbalances), and Parallel search on the internet (speculative parallelism with stochastic workloads).

- Accuracy
  We have investigated the accuracy of our moment approach using the above synthetic and empirical workloads for various compositions. For synthetic workloads, the prediction error is in the percent range. For empirical workloads, the prediction error depends strongly on the correlation and the unimodality of the distributions.

Furthermore, the prediction accuracy is not sensitive to the variance. In particular, the analysis of branch behavior shows that there is a trade-off between accuracy and compositionality.

- A tool implementation
  We have presented a tool implementation of our moment method with full support for the use of stochastic variables in terms of the first four statistical moments. The tool is coined the PAMELA$^+$ compiler. We have shown that for condition synchronization-only models, the PAMELA$^+$ compiler accurately predicts the execution time distribution at low cost. Using the compiler we have also investigated the prediction accuracy with respect to the application to mutual exclusion.

With respect to related work, our approach distinguishes itself as follows.

Unlike workload distribution representations such as pdf, statistical moments are limited in representing the diversity of a large space of input data sets, such as multimodal workloads. The Pdf-based model expresses the density of workload exactly in terms of summation (discrete type) or integral (continuous type). While the use of pdf-based models is typically computationally intensive, our moment approach provide a low-cost, symbolic performance prediction. Unlike specific distribution models, in many cases, our moment approach does not require a unique distribution shape. Especially for parallel composition, the prediction cost using specific distribution models, such as exponomial distributions, is typically low while our moment approach requires much more computation to find the parameters of GLD. While specific distribution models provides a coarse approximation of the workloads as measured in real programs, our moment approach can fit a wide range of distributions with high accuracy.
Compared to arbitrary distribution models, our moment approach covers all possible mathematical operations corresponding to the various types of program control flows.
Compared to moment-based approaches, our approach is a generalization of approaches using up to the first two moments in the sequential domain only.
The examined branching models have led to a closer understanding of branching behavior in static performance modeling. While other models are limited to Bernoulli behavior (with one parameter), which assumes independent branch invocation, we have introduced the ARP branching model (with two parameters) which accounts for some forms correlation between branch invocations, although the possibility of a high variance error still exists. Unlike pdf-based models, our analysis approach is symbolic which allows algebraic optimization techniques to be applied. Unlike arbitrary distribution models, such as Gumbel's method, our approach can be applied to programs with arbitrary nested parallelism.
Like other moment-based approaches, our analysis has $O(1)$ complexity. However, our analysis is also applicable in the parallel domain.

In summary, our statistical moment approach considers various related aspects of symbolic performance modeling of data-dependent parallel programs. Our analysis technique

is symbolic and general, i.e., yields a closed-form performance model based on SP-DAGs, and applies to arbitrary composition and a wide range of workload distribution, respectively. Our technique combines low-cost analysis complexity by critical path analysis in SP-DAGs with acceptable accurate prediction of the execution time distribution by statistical moments for stochastic workloads. This combination offers a balanced trade-off which is appropriate in the first phase of design. To the best of our knowledge, such a comprehensive approach has not been presented before.

7.2 Recommendations

In this research we have been involved in the following areas: analysis, symbolic modeling, program instrumentation, probability theory, and simulation. However, the research could be extended to further improve the accuracy of predicting the execution time of parallel programs. We give the following recommendations for future work in order to address the current limitations of our work:

- Modeling
  Our moment approach is limited to model unimodal workloads. As multimodal workloads can occur in practice, as we have shown in WATOR, these workloads lead to severe degradation of the prediction accuracy. A better approximation for a multimodal workload may be achieved by combining multiple unimodal workloads. Although our moment approach cannot represent the Laplace distribution, some distributions can approximate the Laplace distribution very well as described in [22]. We have introduced three models for branching behavior. In these models we assume that branch correlation is limited to some sequence of branch invocations due to data dependency. However, some branches also depend on the entire sequence, or behave explicitly as function of the loop bound, which lead to errors in almost half of the branches studied (those branches that were input-data set dependent). In the latter cases, the ARP approach turns out to be very sensitive. From these results, a better branch model should account for correlation function

- Analysis
  We have shown that correlated parameters can degrade the prediction accuracy. Although we have introduced an analysis to include the correlation in terms of covariance, the covariance values are hard to measure in practice. Furthermore, correlation can exist in more than two program parameters so that the analysis becomes even more complicated.
  In the PAMELA$^+$ compiler, the moment approach is applicable to the analysis of condition synchronization (φ) *and* mutual exclusion (ω). Yet, the correlation between φ and ω causes an additional prediction error. Although the correlation can be determined by the compiler, there is no analysis technique yet available to handle this problem.

- Tooling
  An ambitious plan for future work is designing a complete automatic performance

evaluation tool for data-dependent parallel programs. The future tool should provide automatic program profiling (where a persistent, incremental profile is maintained, and possibly fed back into the program in terms of performance annotations), bottleneck diagnosis, parameter optimization, and performance visualization.

Appendix A

The Method of Moments

In this appendix, we present a detailed discussion of the method of moments. In Section A.1, we show that the method of moments can approximate distributions in terms of lower moments. In Section A.2, we show that the method of moments is efficient to approximate a distribution under a certain condition.

A.1 Moments as Characteristics of a Distribution

In general the method of moments does not completely determines a distribution, even when moments of all orders exist. Only under certain conditions a set of moments determine a distribution uniquely. Typically, those conditions are satisfied by most of the distributions encountered in practice. A full discussion of the conditions is given in [104]. For most ordinary purposes, knowledge of the moments is equivalent to knowledge of the distribution, in the sense that it should be possible *theoritically* to exhibit all the properties of the distribution in terms of the moments.

In particular, if moments up to order $r \geq 1$ are identical, we expect that as r tends to infinity, the distributions approach each other. Consequently, we expect that by accounting for the lower moments of two distributions, we bring them to approximate equality. Some mathematical support for this may be derived from the following approach [104].

Proposition A.1 Approximation by R moments
A continuous function in a finite range a to b can be represented in that range by a uniformly convergent series of polynomials in x as follows.

$$f(x) = \sum_{r=0}^{\infty} P_r(x) \tag{A.1}$$

where $P_r(x)$ is of degree r. Suppose we want to approximate $f(x)$ by the *finite* series of powers $\sum_{r=0}^{R} a_r x^r$. The coefficient a_r may be determined by the principle of least squares, i.e. to make

$$\int_a^b \left(f(x) - \sum_{r=0}^{R} a_r x^r \right)^2 \mathrm{d}x \tag{A.2}$$

a minimum by taking the derivative with respect to a_r and equating to zero. We obtain

$$2\int_a^b \left(f(x) - \sum_{r=0}^{R} a_r x^r \right) x^i \mathrm{d}x = 0 \tag{A.3}$$

or

$$\int_a^b x^i f(x)\mathrm{d}x = \mu_i' = \int_a^b \sum_{r=0}^{R} a_r x^{r+i}\mathrm{d}x. \tag{A.4}$$

If two distributions have equal moments up to $R \geq 1$, they must have the same least-squares approximation. The coefficient of a_r are determined by the moments in Eq. (A.4). Furthermore, if the distribution $f_1(x)$ differs from $\sum_{r=0}^{R} a_r x^r$ by ϵ_1 and $f_2(x)$ by ϵ_2, then $f_1(x)$ differs from $f_2(x)$ by not more than $\epsilon_1 + \epsilon_2$.

□

Thus, distributions that have equal lower moments up to R will be approximations to one another. Hence, we can approximate a distribution by finding another distribution of known form that has the same lower moments. In practice, this approximation often turn out to be remarkably good, even when only the first two, three or four moments are equated. In our approach, we use the first four moments.

A.2 Efficiency of the Method of Moments

In Section A.1, we focus on the properties of populations only and no description about the reliability of the estimation. If the observations are a *sample* from a population, fitting by moments provides the efficient estimators of the unknown parameters as follows [104].

Proposition A.2 Efficiency of the method of bounded moments
Consider a distribution dependent on $R = 4$ parameters. Note that our approach uses the first four moments. If the maximum-likelihood (ML) estimators of these parameters are to be obtained in terms of linear functions of the moments, we obtain

$$\frac{\partial \log L}{\partial \theta_r} = a_0 + a_1 \sum x + a_2 \sum x^2 + a_3 \sum x^3 + a_4 \sum x^4, \qquad r = 1, \ldots, 4, \tag{A.5}$$

and consequently

$$f(x \mid \theta_1, \ldots, \theta_4) = exp(b_0 + b_1 x + b_2 x^2 + b_3 x^3 + b_4 x^4), \tag{A.6}$$

where the bs depend on the θs. This is the most general form for which the method of moments gives ML estimators. The bs are conditioned by the requirement that $f(x)$ is a valid pdf.

Without loss of generality we may take $b_1 = 0$. If, then, b_3 and b_4 are zero, the distribution is normal and the method of moments is efficient. In other cases, Eq. (A.6) yield an approximation. For example, consider

$$\frac{\partial \log L}{\partial x} = 2b_2 x + 3b_3 x^2 + 4b_4 x^3. \tag{A.7}$$

If b_3 and $b4$ are small, this is approximately

$$\frac{\partial \log L}{\partial x} = \frac{2b_2 x}{1 - \frac{3b_3}{2b_2}x - \frac{2b_4}{b_2}x^2}. \tag{A.8}$$

Only when b_3 and b_4 are small compared with b_2, we can expect the method of moments to give estimators of high efficiency.

□

Appendix B

Task Compositions

In this appendix we represent sequential and parallel compositions in terms of a Directed Acyclic Graph (DAG), with nodes representing tasks, and edges representing task inter-dependencies. Note that a conditional composition cannot be represented by DAGs. For this composition, we use `if (cond) ... else ...` (see Eq. (3.53)) and `switch` (see Eq. (3.55)).

We distinguish DAGs in SP-DAG and non SP-DAG as shown in Figure B.1. Since the analysis for non SP-DAGs is complex, next we consider SP-DAGs only. An SP-DAG is a DAG which can be reduced to a single node using series-parallel reduction. Series and parallel reductions for a composition with N tasks are shown in Figures B.2 and B.3, respectively. The constituting tasks have workloads X_i and the reduced task has workload Y.

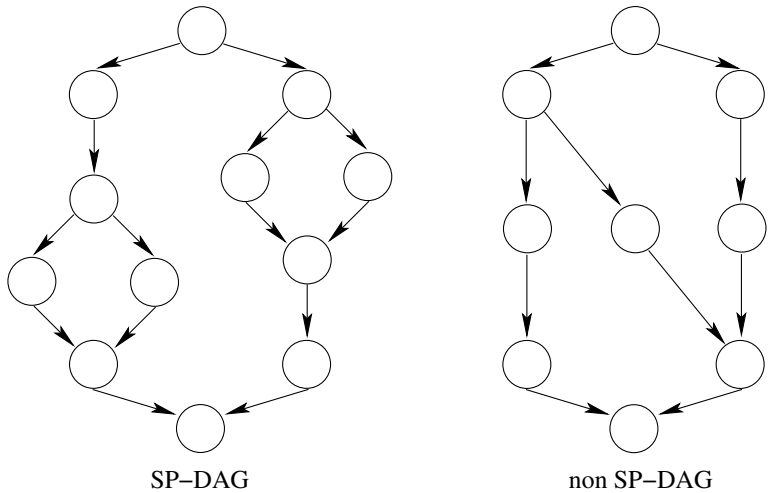

Figure B.1: SP-DAG (left) and non SP-DAG (right)

A sequential composition with N tasks is shown in Figure B.2 (top). The tasks are processed in a strict sequence, from X_1 to X_N. If the tasks are iid (N-ary sequential composition), the associated moment reduction is given by Eq. (3.38). If $N = 2$ and the tasks are different (binary sequential composition), the associated moment reduction is gicen by Eq. (3.33).

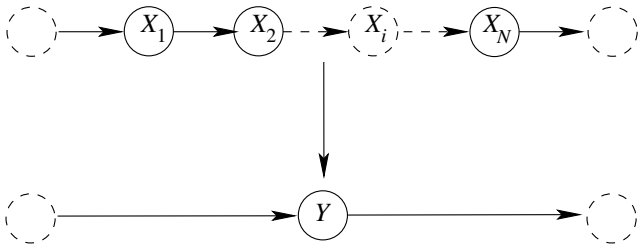

Figure B.2: Series reduction of a sequential composition with N tasks

A parallel composition composition with N tasks is shown in Figure B.3 (left). The tasks are running in parallel without intermediate form of of synchronization. If the tasks are iid (N-ary parallel composition), the associated moment reduction is given by Eq. (5.18). If $N = 2$ and the tasks are different (binary parallel composition), the associated moment reduction is given by Eq. (5.35).

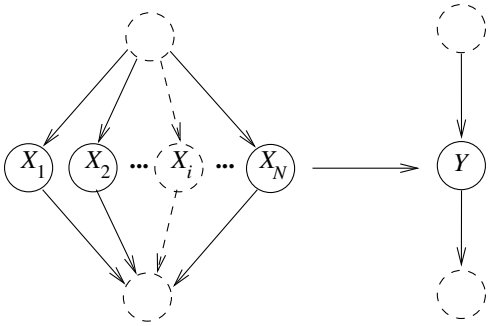

Figure B.3: Parallel reduction of a parallel composition with N tasks

Appendix C

Derivations

In this appendix we provide the full derivations of the four moments for sequential and conditional compositions.

C.1 Sequential Composition

C.1.1 Binary Sequential Composition

Recall the output moments of the binary sequential composition given in Eq. (3.33).

$$\mathsf{E}[Y^r] = \sum_{j=0}^{r} \binom{r}{j} \mathsf{E}[X_1^j]\mathsf{E}[X_2^{r-j}]. \tag{C.1}$$

From Eq. (C.1) we derive the first four moments of Y as follows.

Mean and Variance

For $r = 1$, it follows

$$\begin{aligned} \mathsf{E}[Y] &= \binom{1}{0}\mathsf{E}[X_1^0]\mathsf{E}[X_2] + \binom{1}{1}\mathsf{E}[X_1]\mathsf{E}[X_2^0] \\ &= \mathsf{E}[X_2] + \mathsf{E}[X_1] \\ &= \mathsf{E}[X_1] + \mathsf{E}[X_2]. \end{aligned} \tag{C.2}$$

For $r = 2$, it follows

$$\begin{aligned} \mathsf{E}[Y^2] &= \binom{2}{0}\mathsf{E}[X_1^0]\mathsf{E}[X_2^2] + \binom{2}{1}\mathsf{E}[X_1]\mathsf{E}[X_2] + \binom{2}{2}\mathsf{E}[X_1^2]\mathsf{E}[X_2^0] \\ &= \mathsf{E}[X_2^2] + 2\mathsf{E}[X_1]\mathsf{E}[X_2] + \mathsf{E}[X_1^2]. \end{aligned} \tag{C.3}$$

From Eqs. (C.2) and (C.3) the variance of Y is given by

$$\begin{aligned} \mathsf{Var}[Y] &= \mathsf{E}[Y^2] - \mathsf{E}[Y]^2 \\ &= \mathsf{E}[X_2^2] + 2\mathsf{E}[X_1]\mathsf{E}[X_2] + \mathsf{E}[X_1^2] - (\mathsf{E}[X_1] + \mathsf{E}[X_2])^2 \end{aligned}$$

$$
\begin{aligned}
&= \mathsf{E}[X_2^2] + 2\mathsf{E}[X_1]\mathsf{E}[X_2] + \mathsf{E}[X_1^2] - (\mathsf{E}[X_1]^2 + 2\mathsf{E}[X_1]\mathsf{E}[X_2] + \mathsf{E}[X_2]^2) \\
&= \mathsf{E}[X_1^2] - \mathsf{E}[X_1]^2 + \mathsf{E}[X_2^2] - \mathsf{E}[X_2]^2 \\
&= \mathsf{Var}[X_1] + \mathsf{Var}[X_2]. \qquad \text{(C.4)}
\end{aligned}
$$

Skewness and Kurtosis

For $r = 3$, it follows

$$
\begin{aligned}
\mathsf{E}[Y^3] &= \binom{3}{0}\mathsf{E}[X_1^0]\mathsf{E}[X_2^3] + \binom{3}{1}\mathsf{E}[X_1]\mathsf{E}[X_3^2] + \binom{3}{2}\mathsf{E}[X_1^2]\mathsf{E}[X_2] + \binom{3}{3}\mathsf{E}[X_1^3]\mathsf{E}[X_2^0] \\
&= \mathsf{E}[X_2^3] + 3\mathsf{E}[X_1]\mathsf{E}[X_2^2] + 3\mathsf{E}[X_1^2]\mathsf{E}[X_2] + \mathsf{E}[X_1^3] \qquad \text{(C.5)}
\end{aligned}
$$

From Eqs. (C.2), (C.3) and (C.5) the skewness of Y is given by

$$
\begin{aligned}
\mathsf{Skw}[Y] &= (\mathsf{E}[Y^3] - 3\mathsf{E}[Y^2]\mathsf{E}[Y]^2 + 2\mathsf{E}[Y]^3)/\mathsf{Std}[Y]^3 \\
&= (\mathsf{E}[X_2^3] + 3\mathsf{E}[X_1]\mathsf{E}[X_2^2] + 3\mathsf{E}[X_1^2]\mathsf{E}[X_2] + \mathsf{E}[X_1^3] \\
&\quad - 3(\mathsf{E}[X_2^2] + 2\mathsf{E}[X_1]\mathsf{E}[X_2] + \mathsf{E}[X_1^2])(\mathsf{E}[X_1] + \mathsf{E}[X_2])^2 \\
&\quad + 2(\mathsf{E}[X_1] + \mathsf{E}[X_2]))/\mathsf{Std}[Y]^3 \\
&= (\mathsf{E}[X_2^3] + \mathsf{E}[X_1^3] - 3\mathsf{E}[X_1](2\mathsf{E}[X_1]\mathsf{E}[X_2] + \mathsf{E}[X_1^2]). \qquad \text{(C.6)}
\end{aligned}
$$

For $r = 4$, it follows

$$
\begin{aligned}
\mathsf{E}[Y^4] &= \binom{4}{0}\mathsf{E}[X_1^0]\mathsf{E}[X_2^4] + \binom{4}{1}\mathsf{E}[X_1]\mathsf{E}[X_3^3] + \binom{4}{2}\mathsf{E}[X_1^2]\mathsf{E}[X_2] \\
&\quad + \binom{4}{3}\mathsf{E}[X_1^3]\mathsf{E}[X_2^0] + \binom{4}{4}\mathsf{E}[X_1^4]\mathsf{E}[X_2^0] \\
&= \mathsf{E}[X_2^4] + 4\mathsf{E}[X_1]\mathsf{E}[X_2^3] + 6\mathsf{E}[X_1^2]\mathsf{E}[X_2]^2 + 4\mathsf{E}[X_1^3]\mathsf{E}[X_2] + \mathsf{E}[X_1^4] \qquad \text{(C.7)}
\end{aligned}
$$

From Eqs. (C.2), (C.3), (C.5) and (C.7) the kurtosis of Y is given by

$$
\begin{aligned}
\mathsf{Kur}[Y] &= (\mathsf{E}[Y^3] - 3\mathsf{E}[Y^2]\mathsf{E}[Y]^2 + 2\mathsf{E}[Y]^3)/\mathsf{Std}[Y]^3 \\
&= (\mathsf{E}[X_2^3] + 3\mathsf{E}[X_1]\mathsf{E}[X_2^2] + 3\mathsf{E}[X_1^2]\mathsf{E}[X_2] + \mathsf{E}[X_1^3] \\
&\quad - 3(\mathsf{E}[X_2^2] + 2\mathsf{E}[X_1]\mathsf{E}[X_2] + \mathsf{E}[X_1^2])(\mathsf{E}[X_1] + \mathsf{E}[X_2])^2 \\
&\quad + 2(\mathsf{E}[X_1] + \mathsf{E}[X_2]))/\mathsf{Std}[Y]^3 \\
&= (\mathsf{E}[X_2^3] + \mathsf{E}[X_1^3] - 3\mathsf{E}[X_1](2\mathsf{E}[X_1]\mathsf{E}[X_2] + \mathsf{E}[X_1^2]). \qquad \text{(C.8)}
\end{aligned}
$$

C.1.2 N-ary Sequential Composition

This section presents the derivation of the moments of sums of random number N of iid random variables

$$
Y = \sum_{i=1}^{N} X_i, \qquad \text{(C.9)}
$$

where N is assumed to be a random variable that is independent of the X_i's.

Since the thesis deals with determining the distribution of random variables where random sums as given in Eq. (C.9) occurs, it is extremely important to evaluate the

various parameters of the distribution such as the mean, variance, skewness and kurtosis. In Section C.1.2 the mean and variance of Y are derived [58]. In addition, in this section we extend the existing analysis for the skewness and kurtosis of Y.

Mean and Variance

The mean of Y is found readily by using conditional expectation:

$$\begin{aligned} \mathsf{E}[Y] &= \mathsf{E}[\mathsf{E}[Y|N]] \\ &= \mathsf{E}[N\mathsf{E}[X]] \\ &= \mathsf{E}[N]\mathsf{E}[X]. \end{aligned} \tag{C.10}$$

The second equality follows from the fact that

$$\mathsf{E}[Y|N=n] = \mathsf{E}[\sum_{k=1}^{n} X_i] = n\mathsf{E}[X],$$

so

$$\mathsf{E}[e^{tY}|N] = M_X(t)^N.$$

Therefore

$$\begin{aligned} M_Y(t) &= \mathsf{E}[\mathsf{E}[e^{tY}|N]] \\ &= \mathsf{E}[M_X(t)^N] \\ &= \mathsf{E}[z^N]\Big|_{z=M_X(t)} \\ &= G_N(M_X(t)). \end{aligned} \tag{C.11}$$

Thus, the characteristic function of Y is found by evaluating the generating function of N at $z = M_X(t)$. If N is a continuous random variable, then similarly we can show that

$$M_Y(t) = \mathsf{E}[M_X(t)^N] = M_N(M_X(t)), \tag{C.12}$$

where $M_N(t)$ is the mgf of N. Thus N is not necessary a discrete random variable. This result is very important because Eq. (C.12) also gives the representation of a conditional composition in the transform domain *irrespective* of the Bernoulli assumption. Next in the derivation of formulae we use the mgf rather than the pgf.

To compute $\mathsf{Var}[Y]$, we differentiate $M_Y(t)$ as follows:

$$M_Y^{(1)}(t) = E[NM_X(t)^{N-1}M_X^{(1)}(t)^N], \tag{C.13}$$

$$M_Y^{(2)}(t) = E[N(N-1)M_X(t)^{N-2}M_X^{'}(t)^2 + NM_X(t)^{N-1}M_X^{''}(t)]. \tag{C.14}$$

Evaluating at $t = 0$ gives

$$\mathsf{E}[Y] = \mathsf{E}[N\mathsf{E}[X]] = \mathsf{E}[N]\mathsf{E}[X] \quad \text{(cf. Eq. C.10)}, \tag{C.15}$$

$$\begin{aligned} \mathsf{E}[Y^2] &= \mathsf{E}[N(N-1)E[X]^2 + N\mathsf{E}[X^2]] \\ &= \mathsf{E}[N]\mathsf{Var}[X] + \mathsf{E}[N^2]\mathsf{E}[X]^2. \end{aligned} \tag{C.16}$$

Hence,

$$\begin{aligned} \mathsf{Var}[Y] &= \mathsf{E}[Y^2] - \mathsf{E}[Y]^2 \\ &= \mathsf{E}[N]\mathsf{Var}[X] + \mathsf{E}[X]^2\mathsf{Var}[N]. \end{aligned} \tag{C.17}$$

Skewness and Kurtosis

In Section C.1.2 we have found the mean and variance of Y. In this section we derive the skewness and kurtosis of Y as function of the central moments of X and N using the mgf.

To obtain the skewness of Y we need to evaluate the third derivative of Y by taking the derivative of Eq. (C.14), we have

$$\begin{aligned} M_Y^{(3)}(t) = \mathsf{E}[&N(N-1)(N-2)M_X(t)^{N-3}(M_X^{(1)}(t))^3 \\ &+ 3N(N-1)M_X(t)^{N-2}M_X^{(1)}(t)M_X^{(2)}(t) \\ &+ NM_X(t)^{N-1}M_X^{(3)}(t)]. \end{aligned} \tag{C.18}$$

Evaluating at $t = 0$ gives

$$\begin{aligned} \mathsf{E}[Y^3] &= \mathsf{E}[N(N-1)(N-2)\mathsf{E}[X]^3 + 3N(N-1)\mathsf{E}[X]\mathsf{E}[X^2] + N\mathsf{E}[X^3]] \\ &= \mathsf{E}[N]\mathsf{Skw}[X]\mathsf{Std}[X]^3 + 3\mathsf{E}[N^2]\mathsf{E}[X]\mathsf{Var}[X] + \mathsf{E}[N^3]\mathsf{E}[X]^3. \end{aligned} \tag{C.19}$$

Using Eq. (3.12) the skewness of Y is found

$$\mathsf{Skw}[Y] = \frac{\mathsf{E}[(Y - \mathsf{E}[Y])^3]}{\mathsf{Std}[Y]^3} \tag{C.20}$$

$$\begin{aligned} &= (\mathsf{E}[N]\mathsf{Skw}[X]\mathsf{Std}[X]^3 + \mathsf{E}[X]^3\mathsf{Skw}[N]\mathsf{Std}[N]^3 \\ &\quad + 3\mathsf{E}[X]\mathsf{Var}[X]\mathsf{Var}[N])/\mathsf{Std}[Y]^3. \end{aligned} \tag{C.21}$$

If N is a constant random variable, then

$$\mathsf{Skw}[Y] = \frac{\mathsf{Skw}[X]}{\sqrt{N}}. \tag{C.22}$$

Eq. (C.22) shows that $\mathsf{Skw}[Y]$ approaches zero as function of squared N. The bigger N, the closer $\mathsf{Skw}[Y]$ to zero. It means that the pdf is more symmetric.

Now, we obtain the kurtosis of Y by taking the fourth derivative of Y

$$\begin{aligned} M_Y^{(4)}(t) = \mathsf{E}[&N(N-1)(N-2)(N-3)M_X(t)^{N-4}(M_X^{(1)}(t))^4 \\ &+ 6N(N-1)(N-2)M_X(t)^{N-3}(M_X^{(1)}(t))^2M_X^{(2)}(t) \\ &+ 4N(N-1)M_X(t)^{N-2}M_X^{(1)}(t)M_X^{(3)}(t) \\ &+ 3N(N-1)M_X(t)^{N-2}(M_X^{(2)}(t))^2 \\ &+ NM_X(t)^{N-1}M_X^{(4)}(t). \end{aligned} \tag{C.23}$$

Evaluating at $t = 0$ gives

$$\begin{aligned} \mathsf{E}[Y^4] &= \mathsf{E}[N(N-1)(N-2)(N-3)\mathsf{E}[X]^4 + 6N(N-1)(N-2)\mathsf{E}[X]^2\mathsf{E}[X^2] \\ &\quad + 4N(N-1)\mathsf{E}[X]\mathsf{E}[X^3] + 3N(N-1)\mathsf{E}[X^2]^2 + N\mathsf{E}[X^4]] \\ &= \mathsf{E}[N]\mathsf{Var}[X]^2\mathsf{Kur}[X] + 4\mathsf{E}[N^2]\mathsf{E}[X]\mathsf{Skw}[X]\mathsf{Std}[X]^3 \\ &\quad + 3(\mathsf{E}[N] + \mathsf{E}[N^2])\mathsf{Var}[X]^2 + 6\mathsf{E}[N^3]\mathsf{E}[X]^2\mathsf{Var}[X] + \mathsf{E}[N^4]\mathsf{E}[X]^4. \end{aligned} \tag{C.24}$$

Substituting above expression to Eq. (3.13) we have

$$\begin{aligned}\mathsf{Kur}[Y] = (&\mathsf{E}[N]\mathsf{Kur}[X]\mathsf{Var}[X]^2 + \mathsf{E}[X]^4\mathsf{Kur}[N]\mathsf{Var}[N]^2 \\ &+ 6\mathsf{E}[X]^2\mathsf{Var}[X]\mathsf{Skw}[N]\mathsf{Std}[N]^3 + 4\mathsf{E}[X]\mathsf{Var}[N]\mathsf{Skw}[X]\mathsf{Std}[X]^3 \\ &+ 3\mathsf{Var}[X]^2\mathsf{Var}[N])/\mathsf{Var}[Y]^2. \end{aligned} \tag{C.25}$$

If N is a constant random variable, then Eq. (C.25) is given by

$$\mathsf{Kur}[Y] = \frac{\mathsf{Kur}[X]}{N}. \tag{C.26}$$

The greater N, the closer $\mathsf{Kur}[Y]$ to 0. Note that this value is relative to the kurtosis of the normal distribution.

C.2 Conditional Composition

C.2.1 Binary Conditional Composition

We derive the formulae for binary conditional composition.

$$\mathsf{E}[Y] = \mathsf{E}[P]\mathsf{E}[X_1] + \mathsf{E}[Q]\mathsf{E}[X_2], \tag{C.27}$$

$$\begin{aligned}\mathsf{Var}[Y] &= \mathsf{E}[Y^2] - \mathsf{E}[Y]^2 \\ &= \mathsf{E}[P]\mathsf{Var}[X_1] + \mathsf{E}[P^2]\mathsf{E}[X_1]^2 + \mathsf{E}[Q]\mathsf{Var}[X_2] + \mathsf{E}[Q^2]\mathsf{E}[X_2]^2 \\ &\quad - (\mathsf{E}[P]\mathsf{E}[X_1] + \mathsf{E}[Q]\mathsf{E}[X_2])^2 \\ &= \mathsf{E}[P]\mathsf{Var}[X_1] + \mathsf{Var}[P]\mathsf{E}[X_1]^2 + \mathsf{E}[Q]\mathsf{Var}[X_2] + \mathsf{Var}[Q]\mathsf{E}[X_2]^2 \\ &\quad -2\mathsf{E}[P]\mathsf{E}[Q]\mathsf{E}[X_1]\mathsf{E}[X_2], \end{aligned} \tag{C.28}$$

$$\begin{aligned}\mathsf{Skw}[Y] &= \mathsf{E}[Y^3] - 3\mathsf{E}[Y]\mathsf{E}[Y^2] + 2\mathsf{E}[Y]^3 \\ &= \mathsf{E}[P]\mathsf{Skw}[X_1]\mathsf{Std}[X_1]^3 + 3\mathsf{E}[P^2]\mathsf{E}[X_1]\mathsf{Var}[X_1] + \mathsf{E}[P^3]\mathsf{E}[X_1]^3 \\ &\quad + \mathsf{E}[Q]\mathsf{Skw}[X_2]\mathsf{Std}[X_2]^3 + 3\mathsf{E}[Q^2]\mathsf{E}[X_2]\mathsf{Var}[X_2] + \mathsf{E}[Q^3]\mathsf{E}[X_2]^3 \\ &\quad - 3(\mathsf{E}[P]\mathsf{E}[X_1] + \mathsf{E}[Q]\mathsf{E}[X_2])(\mathsf{E}[P]\mathsf{Var}[X_1] + \mathsf{E}[P^2]\mathsf{E}[X_1]^2 \\ &\quad + \mathsf{E}[Q]\mathsf{Var}[X_1] + \mathsf{E}[Q^2]\mathsf{E}[X_2]^2) + 2(\mathsf{E}[P]\mathsf{E}[X_1] + \mathsf{E}[Q]\mathsf{E}[X_2])^3 \\ &= \mathsf{E}[P]\mathsf{Skw}[X_1]\mathsf{Std}[X_1]^3 + 3\mathsf{E}[P^2]\mathsf{E}[X_1]\mathsf{Var}[X_1] + \mathsf{E}[P^3]\mathsf{E}[X_1]^3 \\ &\quad + \mathsf{E}[Q]\mathsf{Skw}[X_2]\mathsf{Std}[X_2]^3 + 3\mathsf{E}[Q^2]\mathsf{E}[X_2]\mathsf{Var}[X_2] + \mathsf{E}[Q^3]\mathsf{E}[X_2]^3 \\ &\quad - 3(\mathsf{E}[P]\mathsf{E}[X_1] + \mathsf{E}[Q]\mathsf{E}[X_2])(\mathsf{E}[P]\mathsf{Var}[X_1] + \mathsf{E}[P^2]\mathsf{E}[X_1]^2 \\ &\quad + \mathsf{E}[Q]\mathsf{Var}[X_2] + \mathsf{E}[Q^2]\mathsf{E}[X_2]^2) \\ &\quad + 2(\mathsf{E}[P]^3\mathsf{E}[X_1]^3 + 3\mathsf{E}[P]^2\mathsf{E}[X_1]^2\mathsf{E}[Q]\mathsf{E}[X_2] \\ &\quad + 3\mathsf{E}[P]\mathsf{E}[X_1]\mathsf{E}[Q]^2\mathsf{E}[X_2]^2 + \mathsf{E}[Q]^3\mathsf{E}[X_2]^3) \\ &= \mathsf{E}[P]\mathsf{Skw}[X_1]\mathsf{Std}[X_1]^3 + 3\mathsf{E}[P^2]\mathsf{E}[X_1]\mathsf{Var}[X_1] + \mathsf{E}[X_1]^3\mathsf{Skw}[P]\mathsf{Std}[P]^3 \\ &\quad + \mathsf{E}[Q]\mathsf{Skw}[X_2]\mathsf{Std}[X_2]^3 + 3\mathsf{E}[Q^2]\mathsf{E}[X_2]\mathsf{Var}[X_2] + \mathsf{E}[X_2]^3\mathsf{Skw}[Q]\mathsf{Std}[Q]^3 \end{aligned}$$

$$
\begin{aligned}
&- 3\mathsf{E}[P]\mathsf{E}[X_1](\mathsf{E}[P]\mathsf{Var}[X_1] + \mathsf{E}[Q]\mathsf{Var}[X_2] + \mathsf{E}[Q^2]\mathsf{E}[X_2]^2) \\
&- 3\mathsf{E}[Q]\mathsf{E}[X_2](\mathsf{E}[P]\mathsf{Var}[X_1] + \mathsf{E}[P^2]\mathsf{E}[X_1]^2 + \mathsf{E}[Q]\mathsf{Var}[X_2]) \\
&+ 2(3\mathsf{E}[P]^2\mathsf{E}[X_1]^2\mathsf{E}[Q]\mathsf{E}[X_2] + 3\mathsf{E}[P]\mathsf{E}[X_1]\mathsf{E}[Q]^2\mathsf{E}[X_2]^2) \\
&= \mathsf{E}[P]\mathsf{Skw}[X_1]\mathsf{Std}[X_1]^3 + \mathsf{E}[X_1]^3\mathsf{Skw}[P]\mathsf{Std}[P]^3 + 3\mathsf{Var}[P]\mathsf{E}[X_1]\mathsf{Var}[X_1] \\
&+ \mathsf{E}[Q]\mathsf{Skw}[X_2]\mathsf{Std}[X_2]^3 + \mathsf{E}[X_2]^3\mathsf{Skw}[Q]\mathsf{Std}[Q]^3 + 3\mathsf{Var}[Q]\mathsf{E}[X_2]\mathsf{Var}[X_2] \\
&- 3\mathsf{E}[P]\mathsf{E}[X_1](\mathsf{E}[Q]\mathsf{Var}[X_2] + \mathsf{E}[Q^2]\mathsf{E}[X_2]^2) \\
&- 3\mathsf{E}[Q]\mathsf{E}[X_2](\mathsf{E}[P]\mathsf{Var}[X_1] + \mathsf{E}[P^2]\mathsf{E}[X_1]^2) \\
&+ 2(3\mathsf{E}[P]^2\mathsf{E}[X_1]^2\mathsf{E}[Q]\mathsf{E}[X_2] + 3\mathsf{E}[P]\mathsf{E}[X_1]\mathsf{E}[Q]^2\mathsf{E}[X_2]^2) \\
&= \mathsf{E}[P]\mathsf{Skw}[X_1]\mathsf{Std}[X_1]^3 + \mathsf{E}[X_1]^3\mathsf{Skw}[P]\mathsf{Std}[P]^3 + 3\mathsf{Var}[P]\mathsf{E}[X_1]\mathsf{Var}[X_1] \\
&+ \mathsf{E}[Q]\mathsf{Skw}[X_2]\mathsf{Std}[X_2]^3 + \mathsf{E}[X_2]^3\mathsf{Skw}[Q]\mathsf{Std}[Q]^3 + 3\mathsf{Var}[Q]\mathsf{E}[X_2]\mathsf{Var}[X_2] \\
&- 3\mathsf{E}[P]\mathsf{E}[X_1](\mathsf{E}[Q]\mathsf{Var}[X_2] + \mathsf{Var}[Q]\mathsf{E}[X_2]^2 + \mathsf{E}[Q]^2\mathsf{E}[X_2]^2) \\
&- 3\mathsf{E}[Q]\mathsf{E}[X_2](\mathsf{E}[P]\mathsf{Var}[X_1] + \mathsf{Var}[P]\mathsf{E}[X_1]^2 + \mathsf{E}[P]^2\mathsf{E}[X_1]^2) \\
&+ 2(3\mathsf{E}[P]^2\mathsf{E}[X_1]^2\mathsf{E}[Q]\mathsf{E}[X_2] + 3\mathsf{E}[P]\mathsf{E}[X_1]\mathsf{E}[Q]^2\mathsf{E}[X_2]^2) \\
&= \mathsf{E}[P]\mathsf{Skw}[X_1]\mathsf{Std}[X_1]^3 + \mathsf{E}[X_1]^3\mathsf{Skw}[P]\mathsf{Std}[P]^3 + 3\mathsf{Var}[P]\mathsf{E}[X_1]\mathsf{Var}[X_1] \\
&+ \mathsf{E}[Q]\mathsf{Skw}[X_2]\mathsf{Std}[X_2]^3 + \mathsf{E}[X_2]^3\mathsf{Skw}[Q]\mathsf{Std}[Q]^3 + 3\mathsf{Var}[Q]\mathsf{E}[X_2]\mathsf{Var}[X_2] \\
&- 3\mathsf{E}[P]\mathsf{E}[X_1](\mathsf{E}[Q]\mathsf{Var}[X_2] + \mathsf{Var}[Q]\mathsf{E}[X_2]^2) \\
&- 3\mathsf{E}[Q]\mathsf{E}[X_2](\mathsf{E}[P]\mathsf{Var}[X_1] + \mathsf{Var}[P]\mathsf{E}[X_1]^2) \\
&+ 3\mathsf{E}[P]^2\mathsf{E}[X_1]^2\mathsf{E}[Q]\mathsf{E}[X_2] + 3\mathsf{E}[P]\mathsf{E}[X_1]\mathsf{E}[Q]^2\mathsf{E}[X_2]^2 \\
&= \mathsf{E}[P]\mathsf{Skw}[X_1]\mathsf{Std}[X_1]^3 + \mathsf{E}[X_1]^3\mathsf{Skw}[P]\mathsf{Std}[P]^3 + 3\mathsf{Var}[P]\mathsf{E}[X_1]\mathsf{Var}[X_1] \\
&+ \mathsf{E}[Q]\mathsf{Skw}[X_2]\mathsf{Std}[X_2]^3 + \mathsf{E}[X_2]^3\mathsf{Skw}[Q]\mathsf{Std}[Q]^3 + 3\mathsf{Var}[Q]\mathsf{E}[X_2]\mathsf{Var}[X_2] \\
&- 3\mathsf{E}[P]\mathsf{E}[X_1](\mathsf{E}[Q]\mathsf{Var}[X_2] + (\mathsf{Var}[Q] - \mathsf{E}[Q]^2)\mathsf{E}[X_2]^2) \\
&- 3\mathsf{E}[Q]\mathsf{E}[X_2](\mathsf{E}[P]\mathsf{Var}[X_1] + (\mathsf{Var}[P] - \mathsf{E}[P]^2)\mathsf{E}[X_1]^2). \qquad \text{(C.29)}
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{Kur}[Y]\mathsf{Var}[Y]^2 &= \mathsf{E}[Y^4] - 4\mathsf{E}[Y]\mathsf{E}[Y^3] + 6\mathsf{E}[Y]^2\mathsf{E}[Y^2] - 3\mathsf{E}[Y]^4 - 3\mathsf{Var}[Y]^2 \\
&= \mathsf{E}[Y^4] - 4\mathsf{E}[Y](\mathsf{Skw}[Y]\mathsf{Std}[Y]^3 + 3\mathsf{E}[Y]\mathsf{E}[Y^2] - 2\mathsf{E}[Y]^3) \\
&\quad + 6\mathsf{E}[Y]^2\mathsf{E}[Y^2] - 3\mathsf{E}[Y]^4 - 3\mathsf{Var}[Y]^2 \qquad \text{(C.30)} \\
&= \mathsf{E}[Y^4] - 4\mathsf{E}[Y]\mathsf{Skw}[Y]\mathsf{Std}[Y]^3 - 6\mathsf{E}[Y]^2\mathsf{E}[Y^2] + 5\mathsf{E}[Y]^4 - 3\mathsf{Var}[Y]^2 \\
&= \mathsf{E}[Y^4] - 4\mathsf{E}[Y]\mathsf{Skw}[Y]\mathsf{Std}[Y]^3 - 6\mathsf{E}[Y]^2(\mathsf{Var}[Y] + \mathsf{E}[Y]^2) + 5\mathsf{E}[Y]^4 - 3\mathsf{Var}[Y]^2 \\
&= \mathsf{E}[Y^4] - 4\mathsf{E}[Y]\mathsf{Skw}[Y]\mathsf{Std}[Y]^3 - 6\mathsf{E}[Y]^2\mathsf{Var}[Y] - \mathsf{E}[Y]^4 - 3\mathsf{Var}[Y]^2. \qquad \text{(C.31)}
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{Kur}[Y] &= \mathsf{E}[P]\mathsf{Kur}[X_1]\mathsf{Var}[X_1]^2 + 4\mathsf{E}[P^2]\mathsf{E}[X_1]\mathsf{Skw}[X_1]\mathsf{Std}[X_1]^3 \\
&\quad + 3(\mathsf{E}[P] + \mathsf{E}[P^2])\mathsf{Var}[X_1]^2 + 6\mathsf{E}[P^3]\mathsf{E}[X_1]^2\mathsf{Var}[X_1] + \ \mathsf{E}[P^4]\mathsf{E}[X_1]^4 \\
&\quad + \mathsf{E}[Q]\mathsf{Kur}[X_2]\mathsf{Var}[X_2]^2 + 4\mathsf{E}[Q^2]\mathsf{E}[X_2]\mathsf{Skw}[X_2]\mathsf{Std}[X_2]^3 \\
&\quad + 3(\mathsf{E}[Q] + \mathsf{E}[Q^2])\mathsf{Var}[X_2]^2 + 6\mathsf{E}[Q^3]\mathsf{E}[X_2]^2\mathsf{Var}[X_2] + \ \mathsf{E}[Q^4]\mathsf{E}[X_2]^4 \\
&= \mathsf{E}[P]\mathsf{Kur}[X_1]\mathsf{Var}[X_1]^2 + \mathsf{E}[Q]\mathsf{Kur}[X_2]\mathsf{Var}[X_2]^2 \\
&\quad + 4\mathsf{E}[P^2]\mathsf{E}[X_1]\mathsf{Skw}[X_1]\mathsf{Std}[X_1]^3 + 4\mathsf{E}[Q^2]\mathsf{E}[X_2]\mathsf{Skw}[X_2]\mathsf{Std}[X_2]^3
\end{aligned}
$$

$$
\begin{aligned}
&+ 3(\mathsf{E}[P] + \mathsf{E}[P^2])\mathsf{Var}[X_1]^2 + 6\mathsf{E}[P^3]\mathsf{E}[X_1]^2\mathsf{Var}[X_1] + \ \mathsf{E}[P^4]\mathsf{E}[X_1]^4 \\
&+ 3(\mathsf{E}[Q] + \mathsf{E}[Q^2])\mathsf{Var}[X_2]^2 + 6\mathsf{E}[Q^3]\mathsf{E}[X_2]^2\mathsf{Var}[X_2] + \ \mathsf{E}[Q^4]\mathsf{E}[X_2]^4 \\
&= \mathsf{E}[P]\mathsf{Kur}[X_1]\mathsf{Var}[X_1]^2 + \mathsf{E}[Q]\mathsf{Kur}[X_2]\mathsf{Var}[X_2]^2 \\
&+ 4\mathsf{E}[P^2]\mathsf{E}[X_1]\mathsf{Skw}[X_1]\mathsf{Std}[X_1]^3 + 4\mathsf{E}[Q^2]\mathsf{E}[X_2]\mathsf{Skw}[X_2]\mathsf{Std}[X_2]^3 \\
&+ \mathsf{E}[X_1]^4(\mathsf{Kur}[P]\mathsf{Var}[P]^2 + 4\mathsf{E}[P]\mathsf{Skw}[P]\mathsf{Std}[P]^3 \\
&+ 6\mathsf{E}[P]^2\mathsf{Var}[P] + \mathsf{E}[P]^4 + 3\mathsf{Var}[P]^2) \\
&+ \mathsf{E}[X_2]^4(\mathsf{Kur}[Q]\mathsf{Var}[Q]^2 + 4\mathsf{E}[Q]\mathsf{Skw}[Q]\mathsf{Std}[Q]^3 \\
&+ 6\mathsf{E}[Q]^2\mathsf{Var}[Q] + \mathsf{E}[Q]^4 + 3\mathsf{Var}[Q]^2) \\
&+ 3(\mathsf{E}[P] + \mathsf{E}[P^2])\mathsf{Var}[X_1]^2 + 6\mathsf{E}[P^3]\mathsf{E}[X_1]^2\mathsf{Var}[X_1] \\
&+ 3(\mathsf{E}[Q] + \mathsf{E}[Q^2])\mathsf{Var}[X_2]^2 + 6\mathsf{E}[Q^3]\mathsf{E}[X_2]^2\mathsf{Var}[X_2] \\
&= \mathsf{E}[P]\mathsf{Kur}[X_1]\mathsf{Var}[X_1]^2 + \mathsf{E}[Q]\mathsf{Kur}[X_2]\mathsf{Var}[X_2]^2 \\
&+ 4\mathsf{E}[P^2]\mathsf{E}[X_1]\mathsf{Skw}[X_1]\mathsf{Std}[X_1]^3 + 4\mathsf{E}[Q^2]\mathsf{E}[X_2]\mathsf{Skw}[X_2]\mathsf{Std}[X_2]^3 \\
&+ \mathsf{E}[X_1]^4(\mathsf{Kur}[P]\mathsf{Var}[P]^2 + 4\mathsf{E}[P]\mathsf{Skw}[P]\mathsf{Std}[P]^3 \\
&+ 6\mathsf{E}[P]^2\mathsf{Var}[P] + \mathsf{E}[P]^4 + 3\mathsf{Var}[P]^2) \\
&+ \mathsf{E}[X_2]^4(\mathsf{Kur}[Q]\mathsf{Var}[Q]^2 + 4\mathsf{E}[Q]\mathsf{Skw}[Q]\mathsf{Std}[Q]^3 \\
&+ 6\mathsf{E}[Q]^2\mathsf{Var}[Q] + \mathsf{E}[Q]^4 + 3\mathsf{Var}[Q]^2) \\
&+ 6\mathsf{E}[X_1]^2\mathsf{Var}[X_1](\mathsf{Skw}[P]\mathsf{Std}[P]^3 + 3\mathsf{E}[P](\mathsf{Var}[P] + \mathsf{E}[P]^2) - 2\mathsf{E}[P]^3) \\
&+ 6\mathsf{E}[X_2]^2\mathsf{Var}[X_2](\mathsf{Skw}[Q]\mathsf{Std}[Q]^3 + 3\mathsf{E}[Q](\mathsf{Var}[Q] + \mathsf{E}[Q]^2) - 2\mathsf{E}[Q]^3) \\
&+ 3(\mathsf{E}[P] + \mathsf{Var}[P] + \mathsf{E}[P]^2)\mathsf{Var}[X_1]^2 + 3(\mathsf{E}[Q] + \mathsf{Var}[Q] + \mathsf{E}[Q]^2)\mathsf{Var}[X_2]^2 \\
&= \mathsf{E}[P]\mathsf{Kur}[X_1]\mathsf{Var}[X_1]^2 + \mathsf{E}[Q]\mathsf{Kur}[X_2]\mathsf{Var}[X_2]^2 \\
&+ 4\mathsf{E}[P^2]\mathsf{E}[X_1]\mathsf{Skw}[X_1]\mathsf{Std}[X_1]^3 + 4\mathsf{E}[Q^2]\mathsf{E}[X_2]\mathsf{Skw}[X_2]\mathsf{Std}[X_2]^3 \\
&+ \mathsf{E}[X_1]^4((\mathsf{Kur}[P] + 3)\mathsf{Var}[P]^2 + 4\mathsf{E}[P]\mathsf{Skw}[P]\mathsf{Std}[P]^3 \\
&+ 6\mathsf{E}[P]^2\mathsf{Var}[P] + \mathsf{E}[P]^4) \\
&+ \mathsf{E}[X_2]^4((\mathsf{Kur}[Q] + 3)\mathsf{Var}[Q]^2 + 4\mathsf{E}[Q]\mathsf{Skw}[Q]\mathsf{Std}[Q]^3 \\
&+ 6\mathsf{E}[Q]^2\mathsf{Var}[Q] + \mathsf{E}[Q]^4) \\
&+ 6\mathsf{E}[X_1]^2\mathsf{Var}[X_1](\mathsf{Skw}[P]\mathsf{Std}[P]^3 + 3\mathsf{E}[P]\mathsf{Var}[P] + \mathsf{E}[P]^3) \\
&+ 6\mathsf{E}[X_2]^2\mathsf{Var}[X_2](\mathsf{Skw}[Q]\mathsf{Std}[Q]^3 + 3\mathsf{E}[Q]\mathsf{Var}[Q] + \mathsf{E}[Q]^3) \\
&+ 3(\mathsf{E}[P] + \mathsf{E}[P]^2 + \mathsf{Var}[P])\mathsf{Var}[X_1]^2 + 3(\mathsf{E}[Q] + \mathsf{E}[Q]^2 + \mathsf{Var}[Q])\mathsf{Var}[X_2]^2.
\end{aligned}
\tag{C.32}
$$

C.2.2 ARP Approach

In this section we derive the moments of branch probability Q_a for the ARP approach expressed in terms of D and U. Recall Eq. (4.13), the rth raw moment of the beta distribution Q_a is given by

$$
\mathsf{E}[Q_a^r] = \frac{\Gamma(a+b)\Gamma(a+r)}{\Gamma(a)\Gamma(a+b+r)} \tag{C.33}
$$

where

$$a = \frac{(\mathsf{E}[D]^2(\mathsf{E}[U] - \mathsf{Var}[U]) + \mathsf{E}[U]^2(\mathsf{E}[D] - \mathsf{Var}[D]))\mathsf{E}[D]}{(\mathsf{E}[D] + \mathsf{E}[U])(\mathsf{E}[D]^2\mathsf{Var}[U] + \mathsf{E}[U]^2\mathsf{Var}[D])} \quad \text{and} \quad b = \frac{\mathsf{E}[U]}{\mathsf{E}[D]}\, a. \tag{C.34}$$

From Eq. (C.33) the moments of Q_a, expressed in terms of a and b, are given by

$$\mathsf{E}[Q_a] = \frac{a}{a+b}, \tag{C.35a}$$

$$\mathsf{Var}[Q_a] = \frac{ab}{(a+b)^2(a+b+1)}, \tag{C.35b}$$

$$\mathsf{Skw}[Q_a] = \frac{2(b-a)\sqrt{a+b+1}}{\sqrt{ab}(a+b+2)}, \tag{C.35c}$$

$$\mathsf{Kur}[Q_a] = \frac{6(a^3 + a^2(1-2b) + b^2(1+b) - 2ab(2+b))}{ab(a+b+2)(a+b+3)} + 3. \tag{C.35d}$$

The moments of Q_a, now expressed in terms of D and U, are obtained by substituting Eq. (C.34) to Eq. (C.35) according to

$$\begin{aligned}
\mathsf{E}[Q_a] &= \frac{a}{a+b} = \frac{1}{1+b/a} \\
&= \frac{1}{1+\mathsf{E}[U]/\mathsf{E}[D]} \\
&= \frac{\mathsf{E}[D]}{\mathsf{E}[D]+\mathsf{E}[U]},
\end{aligned} \tag{C.36a}$$

$$\begin{aligned}
\mathsf{Var}[Q_a] &= \frac{ab}{(a+b)^2(a+b+1)} = \frac{a}{a+b} \quad \frac{b}{a+b} \quad \frac{1}{a+b+1} \\
&= \frac{\mathsf{E}[D]\mathsf{E}[U]}{(\mathsf{E}[D]+\mathsf{E}[U])^2} \\
&\quad \frac{1}{\frac{\mathsf{E}[D]^2(\mathsf{E}[U]-\mathsf{Var}[U])+\mathsf{E}[U]^2(\mathsf{E}[D]-\mathsf{Var}[D])}{(\mathsf{E}[D]+\mathsf{E}[U])(\mathsf{E}[D]^2\mathsf{Var}[U]+\mathsf{E}[U]^2\mathsf{Var}[D])}(\mathsf{E}[D]+\mathsf{E}[U])+1} \\
&= \frac{\mathsf{E}[D]\mathsf{E}[U]}{(\mathsf{E}[D]+\mathsf{E}[U])^2} \\
&\quad \frac{(\mathsf{E}[D]^2\mathsf{Var}[U] + \mathsf{E}[U]^2\mathsf{Var}[D])}{\mathsf{E}[D]^2(\mathsf{E}[U]-\mathsf{Var}[U]) + \mathsf{E}[U]^2(\mathsf{E}[D]-\mathsf{Var}[D]) + (\mathsf{E}[D]^2\mathsf{Var}[U] + \mathsf{E}[U]^2\mathsf{Var}[D])} \\
&= \frac{\mathsf{E}[D]^2\mathsf{Var}[U] + \mathsf{E}[U]^2\mathsf{Var}[D]}{(\mathsf{E}[D]+\mathsf{E}[U])^3},
\end{aligned} \tag{C.36b}$$

$$\begin{aligned}
\mathsf{Skw}[Q_a] &= \frac{2(b-a)\sqrt{a+b+1}}{\sqrt{ab}(a+b+2)} \quad \frac{a+b}{a+b} = \frac{2(b-a)}{a+b} \quad \frac{1}{a+b+2} \quad \frac{(a+b)\sqrt{a+b+1}}{\sqrt{ab}} \\
&= \frac{2(\mathsf{E}[U]/\mathsf{E}[D]-1)}{1+\mathsf{E}[U]/\mathsf{E}[D]}
\end{aligned}$$

$$\frac{1}{\frac{\mathsf{E}[D]^2(\mathsf{E}[U]-\mathsf{Var}[U])+\mathsf{E}[U]^2(\mathsf{E}[D]-\mathsf{Var}[D])}{(\mathsf{E}[D]+\mathsf{E}[U])(\mathsf{E}[D]^2\mathsf{Var}[U]+\mathsf{E}[U]^2\mathsf{Var}[D])}(\mathsf{E}[D]+\mathsf{E}[U])+2} \quad \frac{1}{\mathsf{Std}[Q_a]}$$

$$= \frac{2(\mathsf{E}[U]-\mathsf{E}[D])}{\mathsf{E}[D]+\mathsf{E}[U]}$$

$$\frac{1}{\mathsf{E}[D]^2(\mathsf{E}[U]+\mathsf{Var}[U])+\mathsf{E}[U]^2(\mathsf{E}[D]+\mathsf{Var}[D])} \quad \frac{1}{\mathsf{Std}[Q_a]} \quad \frac{\mathsf{Std}[Q_a]}{\mathsf{Std}[Q_a]}$$

$$= \frac{2(\mathsf{E}[U]-\mathsf{E}[D])}{\mathsf{E}[D]+\mathsf{E}[U]}$$

$$\frac{(\mathsf{E}[D]^2\mathsf{Var}[U]+\mathsf{E}[U]^2\mathsf{Var}[D])}{\mathsf{E}[D]^2(\mathsf{E}[U]+\mathsf{Var}[U])+\mathsf{E}[U]^2(\mathsf{E}[D]+\mathsf{Var}[D])} \quad \frac{\mathsf{Std}[Q_a](\mathsf{E}[D]+\mathsf{E}[U])^3}{\mathsf{E}[D]^2\mathsf{Var}[U]+\mathsf{E}[U]^2\mathsf{Var}[D]}$$

$$= \frac{-2(\mathsf{E}[D]-\mathsf{E}[U])(\mathsf{E}[D]+\mathsf{E}[U])^2\mathsf{Std}[Q_a]}{\mathsf{E}[D]^2(\mathsf{E}[U]+\mathsf{Var}[U])+\mathsf{E}[U]^2(\mathsf{E}[D]+\mathsf{Var}[D])}, \tag{C.36c}$$

$$\mathsf{Kur}[Q_a] = \frac{6(a^3+a^2(1-2b)+b^2(1+b)-2ab(2+b))}{ab(a+b+2)(a+b+3)} + 3$$

$$= \frac{1}{ab}\frac{1}{a+b+2}\frac{1}{a+b+3}$$

$$= \frac{(\mathsf{E}[D]+\mathsf{E}[U])^2(\mathsf{E}[D]^2\mathsf{Var}[U]+\mathsf{E}[U]^2\mathsf{Var}[D])^2}{(\mathsf{E}[D]^2(\mathsf{E}[U]-\mathsf{Var}[U])+\mathsf{E}[U]^2(\mathsf{E}[D]-\mathsf{Var}[D]))^2\mathsf{E}[D]\mathsf{E}[U]}$$

$$\frac{(\mathsf{E}[D]^2\mathsf{Var}[U]+\mathsf{E}[U]^2\mathsf{Var}[D])}{\mathsf{E}[D]^2(\mathsf{E}[U]+\mathsf{Var}[U])+\mathsf{E}[U]^2(\mathsf{E}[D]+\mathsf{Var}[D])}$$

$$\frac{(\mathsf{E}[D]^2\mathsf{Var}[U]+\mathsf{E}[U]^2\mathsf{Var}[D])}{\mathsf{E}[D]^2(\mathsf{E}[U]+2\mathsf{Var}[U])+\mathsf{E}[U]^2(\mathsf{E}[D]+2\mathsf{Var}[D])}$$

$$= 3(\mathsf{E}[D]+\mathsf{E}[U])[(\mathsf{E}[D]+\mathsf{E}[U])\mathsf{E}[D]^2\mathsf{E}[U]^2$$
$$+ (2(\mathsf{E}[D]-\mathsf{E}[U])^2+\mathsf{E}[D]\mathsf{E}[U])(\mathsf{E}[D]^2\mathsf{Var}[U]+\mathsf{E}[U]^2\mathsf{Var}[D])]$$
$$\times [\mathsf{E}[D]^2(\mathsf{E}[U]+2\mathsf{Var}[U])+\mathsf{E}[U]^2(\mathsf{E}[D]+2\mathsf{Var}[D])]^{-1}$$
$$\times [\mathsf{E}[D]^2(\mathsf{E}[U]+\mathsf{Var}[U])+\mathsf{E}[U]^2(\mathsf{E}[D]+\mathsf{Var}[D])]^{-1}. \tag{C.36d}$$

In the rest of this section, we derive the first four moments of P_a when D and U are geometrically distributed with parameter p. The mean and variance of D and U are given by

$$\mathsf{E}[D] = 1/p, \tag{C.37a}$$
$$\mathsf{Var}[D] = (1-p)/p^2, \tag{C.37b}$$
$$\mathsf{E}[U] = 1/(1-p), \tag{C.37c}$$
$$\mathsf{Var}[U] = p/(1-p)^2, \tag{C.37d}$$

respectively. From Eqs. (C.36) and (C.37) we obtain

$$\mathsf{E}[P_a] = 1 - \frac{\mathsf{E}[D]}{\mathsf{E}[D]+\mathsf{E}[U]} = \frac{\mathsf{E}[U]}{\mathsf{E}[D]+\mathsf{E}[U]}$$

$$
\begin{aligned}
&= \frac{1/(1-p)}{1/p + 1/(1-p)} \\
&= \frac{p}{(1-p)+p} \\
&= p, && \text{(C.38a)} \\
\mathsf{Var}[P_a] &= \frac{\mathsf{E}[D]^2\mathsf{Var}[U] + \mathsf{E}[U]^2\mathsf{Var}[D]}{(\mathsf{E}[D] + \mathsf{E}[U])^3} \\
&= \frac{(1/p)^2(p/(1-p)^2) + (1/(1-p))^2((1-p)/p^2)}{(1/p + 1/(1-p))^3} \quad \frac{(p(1-p))^3}{(p(1-p))^3} \\
&= \frac{p^2(1-p) + p(1-p)^2}{(1-p)+p} \\
&= p(1-p)(p + (1-p)) \\
&= p(1-p), && \text{(C.38b)} \\
\mathsf{Skw}[P_a] &= -\frac{-2(\mathsf{E}[D] - \mathsf{E}[U])(\mathsf{E}[D] + \mathsf{E}[U])^2\mathsf{Std}[Q_a]}{\mathsf{E}[D]^2(\mathsf{E}[U] + \mathsf{Var}[U]) + \mathsf{E}[U]^2(\mathsf{E}[D] + \mathsf{Var}[D])} \\
&= \frac{2(1/p - 1/(1-p))(1/p + 1/(1-p))^2(p(1-p))^{1/2}}{(1/p)^2(1/(1-p) + p/(1-p)^2) + (1/(1-p))^2(1/p + (1-p)/p^2)} \\
&\quad \frac{(p(1-p))^3}{(p(1-p))^3} \\
&= \frac{2((1-p) - p)((1-p) + p)^2(p(1-p))^{1/2}}{p((1-p)^2 + p(1-p)) + ((1-p)(p^2 + p(1-p))} \\
&= \frac{2(1-2p)(p(1-p))^{1/2}}{p(1-p) + p(1-p)} \quad \frac{(p(1-p))^{1/2}}{(p(1-p))^{1/2}} \\
&= \frac{2p(1-p)(1-2p)}{2(p(1-p))^{3/2}} \\
&= p(1-p)(1-2p)/(p(1-p))^{3/2}, && \text{(C.38c)} \\
\mathsf{Kur}[P_a] &= 3(\mathsf{E}[D] + \mathsf{E}[U])[(\mathsf{E}[D] + \mathsf{E}[U])\mathsf{E}[D]^2\mathsf{E}[U]^2 \\
&\quad + (2(\mathsf{E}[D] - \mathsf{E}[U])^2 + \mathsf{E}[D]\mathsf{E}[U])(\mathsf{E}[D]^2\mathsf{Var}[U] + \mathsf{E}[U]^2\mathsf{Var}[D])] \\
&\quad \times [\mathsf{E}[D]^2(\mathsf{E}[U] + 2\mathsf{Var}[U]) + \mathsf{E}[U]^2(\mathsf{E}[D] + 2\mathsf{Var}[D])]^{-1} \\
&\quad \times [\mathsf{E}[D]^2(\mathsf{E}[U] + \mathsf{Var}[U]) + \mathsf{E}[U]^2(\mathsf{E}[D] + \mathsf{Var}[D])]^{-1} \\
&= 3(1/p + 1/(1-p))[(1/p + 1/(1-p))(1/p)^2(1/(1-p))^2 \\
&\quad + (2(1/p - 1/(1-p))^2 + (1/p)(1/(1-p))) \\
&\quad \times (1/p^2(p/(1-p)^2) + (1/(1-p))^2(1-p)/p^2)] \\
&\quad \times [1/p^2(1/(1-p) + 2p/(1-p)^2) + 1/(1-p)^2(1/p + 2(1-p)/p^2)]^{-1} \\
&\quad \times [1/p^2(1/(1-p) + p/(1-p)^2) + 1/(1-p)^2(1/p + (1-p)/p^2)]^{-1}
\end{aligned}
$$

$$
\begin{aligned}
&\frac{(p(1-p))^4}{(p(1-p))^4} \\
&= 3((1-p)+p)[(1-p)+p) + (2p(1-p)(1-p-p)^2) + p^2(1-p)^2)(p(1-p))] \\
&\quad \times [p((1-p)^2 + 2p(1-p)) + ((1-p)(p^2 + 2p(1-p))]^{-1} \\
&\quad \times [p((1-p)^2 + p(1-p)) + ((1-p)(p^2 + p(1-p))]^{-1} \\
&= 3[1 + (2p(1-p)(1-2p)^2 + p^2(1-p)^2)p(1-p)] \\
&\quad \times [p(1-p)^2 + p(2-p)(1-p)^2]^{-1} \\
&\quad \times [2p(1-p)]^{-1} \\
&= \frac{3p(1-p)(1-3p+3p^2)}{3(p(1-p))^2} \\
&= p(1-p)(1-3p+3p^2)/(p(1-p))^2 \qquad \text{(C.38d)}
\end{aligned}
$$

Eq. (C.38) shows that $\mathsf{E}[P_a^r] = \mathsf{E}[P_B^r]$ for $r \leq 4$ when D and U are geometrically distributed.

Appendix D

The Gram-Charlier Series of Type A

A general approximation technique of constructing a distribution from moments is the Gram-Charlier series of type A [51]. The series provide an expansion of a distribution in terms of its central moments, the standard normal density, and Hermite polynomials. The pdf of the series is defined by

$$f(x) = \sum_{i=0}^{\infty} c_i H_i(x)\phi(x), \tag{D.1}$$

where $\phi(x)$ is the density function of a standard normal distribution

$$\phi(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right), \tag{D.2}$$

and where $H_i(x)$ is Hermite polynomials of degree i defined by

$$H_i(x) = x^i - \frac{i^{[2]}}{2.1!}x^{i-2} + \frac{i^{[4]}}{2^2.2!}x^{i-4} - \frac{i^{[6]}}{2^3.3!}x^{i-6} + \cdots \tag{D.3}$$

where $i^{[r]} = i(i-1)(i-2)\cdots(i-r+1)$ and by convention $H_0(x) = 1$, and where

$$c_i = \frac{1}{i!}\int_{-\infty}^{\infty} f(x)H_i(x)\mathrm{d}x. \tag{D.4}$$

Substituting the explicit value of $H_i(x)$ from Eq. (D.3) we find

$$c_i = \frac{1}{i!}(\mu_i' - \frac{i^{[2]}}{2.1!}\mu_{i-2}' + \frac{i^{[4]}}{2^2.2!}\mu_{i-4}' - \frac{i^{[6]}}{2^3.3!}\mu_{i-6}' + \cdots). \tag{D.5}$$

In particular, for moments about the mean, the series can then be written as

$$\begin{aligned} f(x) = \phi(x)[1 &+ \frac{1}{2}(\mu_2 - 1)H_2(x) + \frac{1}{6}(\mu_3 - 1)H_3(x) \\ &+ \frac{1}{24}(\mu_4 - 6\mu_2 + 1)H_4(x) + \ldots]. \end{aligned} \tag{D.6}$$

The cdf of the series in Eq. (D.6) can be obtained as follows

$$\begin{aligned} F(x) = \int_{-\infty}^{x} f(t)\mathrm{d}t &= \int_{-\infty}^{x} \sum_{i=0}^{\infty} c_i H_i(t)\phi(x)\mathrm{d}t \\ &= \int_{-\infty}^{x} (\phi(t) + \sum_{i=1}^{\infty} c_i H_i(t)\phi(t))\mathrm{d}t. \\ &= \int_{-\infty}^{x} \phi(t)\mathrm{d}t + \int_{-\infty}^{x} \sum_{i=1}^{\infty} c_i H_i(t)\phi(t)\mathrm{d}t \\ &= \Phi(x) + \sum_{i=1}^{\infty} c_i \int_{-\infty}^{x} H_i(t)\phi(t)\mathrm{d}t \end{aligned} \tag{D.7}$$

where

$$\Phi(x) = \int_{-\infty}^{x} \phi(t)\mathrm{d}t \tag{D.8}$$

which known as the error function in mathematics. Using the fact that

$$H_i(t)\phi(t) = (-\frac{\mathrm{d}}{\mathrm{d}t})^i \phi(t) \tag{D.9}$$

Eq. (D.7) reduces

$$\begin{aligned} F(x) &= \Phi(x) + \sum_{i=1}^{\infty} c_i \int_{-\infty}^{x} (-\frac{\mathrm{d}}{\mathrm{d}x})^i \phi(x)\mathrm{d}x \\ &= \Phi(x) - \sum_{i=1}^{\infty} c_i \left[(-\frac{d}{\mathrm{d}x})^{i-1} \phi(x) \right]_{-\infty}^{x} \\ &= \Phi(x) - \sum_{i=1}^{\infty} c_i \left[H_{i-1}(x)\phi(x) \right]_{-\infty}^{x} \\ &= \Phi(x) - \sum_{i=1}^{\infty} c_i H_{i-1}(x)\phi(x). \end{aligned} \tag{D.10}$$

The series in Eq. (D.6) will converge for every x to $f(x)$ if the following conditions are satisfied

1. $f(x)$ is continuous and has bounded variation on $[-\infty, \infty]$.
2. $\int_{-\infty}^{\infty} f(x) \exp(x^2/4) dx$ is convergent.

Many investigation has been done in neglecting higher moments. These showed that the finite series is useful only in cases of moderate skewness, and in such case a Pearson distribution may be just as good.

From the statistical point of view, however, the important question is not whether an infinite series can represent a density function, but whether a finite number of term can do so to a satisfactory approximation. Two things seem clear:

1. The sum of a finite number of terms of the series may give negative values, particularly near the tails

2. The series may behave irregularly in the sense that the sum of k terms may give a worse fit than the sum of $k-1$ terms.

Next, we will express the moments of distribution function corresponding to the binary max operation using Gram-Charlier series. The following derivation shows that we can express the moments as function of the moments of the two operands explicitly.

Theorem D.1 Given two mutually independent continuous random variables X_1 and X_2 defined by Gram-Charlier series by

$$f_1(x) = \sum_{i=0}^{\infty} c_i^{'} H_i(x)\phi(x), \tag{D.11}$$

and

$$f_2(x) = \sum_{i=0}^{\infty} c_i^{''} H_i(x)\phi(x), \tag{D.12}$$

respectively. Let Y be the maximum of X_1 and X_2

$$Y = \max(X_1, X_2).$$

then the rth moment of Y is given by

$$\mathsf{E}[Y^r] = \sum_{i=0}^{\infty} (c_i^{'} + c_i^{''}) l_{ir}^{'} - \sum_{i=0}^{\infty}\sum_{j=1}^{\infty} (c_i^{''} c_j^{'} + c_i^{'} c_j^{''}) l_{ijr}^{''} \tag{D.13}$$

where $l_{ir}^{'}$ and $l_{ijr}^{''}$ are equal to

$$l_{ir}^{'} = \int_{-\infty}^{\infty} x^r H_i(x)\Phi(x)\phi(x)\mathrm{d}x \tag{D.14}$$

and

$$l_{ijr}^{''} = \int_{-\infty}^{\infty} x^r H_i(x)H_{j-1}(x)\phi^2(x)\mathrm{d}x. \tag{D.15}$$

Proof
The cdf of Y is given by

$$F_Y(x) = F_1(x)F_2(x). \tag{D.16}$$

To evaluate the pdf of Y we evaluate the derivation of $F(x)$ to x

$$\begin{aligned}
f_Y(x) &= \mathrm{d}F(x)/\mathrm{d}x \\
&= F_1(x)f_2(x) + F_2(x)f_1(x) \\
&= (\Phi(x) - \sum_{j=1}^{\infty} c_j^{'} H_{j-1}(x)\phi(x)) \sum_{i=0}^{\infty} c_i^{''} H_i(x)\phi(x) \\
&\quad +(\Phi(x) - \sum_{j=1}^{\infty} c_j^{''} H_{j-1}(x)\phi(x)) \sum_{i=0}^{\infty} c_i^{'} H_i(x)\phi(x) \\
&= \sum_{i=0}^{\infty}(c_i^{'} + c_i^{''}) H_i(x)\Phi(x)\phi(x) \\
&\quad -(\sum_{j=1}^{\infty} c_j^{'} H_{j-1}(x) \sum_{i=0}^{\infty} c_i^{''} H_i(x) + \sum_{j=1}^{\infty} c_j^{''} H_{j-1}(x) \sum_{i=0}^{\infty} c_i^{'} H_i(x))\phi^2(x) \\
&= \sum_{i=0}^{\infty}(c_i^{'} + c_i^{''}) H_i(x)\Phi(x)\phi(x) - \sum_{i=0}^{\infty}\sum_{j=1}^{\infty} (c_i^{''} c_j^{'} + c_i^{'} c_j^{''}) H_i(x)H_{j-1}(x)\phi^2(x).
\end{aligned} \tag{D.17}$$

From Eq. (D.17) the rth moment of X is obtained as follows

$$\begin{aligned}\mathsf{E}[Y^r] &= \int_{-\infty}^{\infty} x^r f(x)\mathrm{d}x \\ &= \sum_{i=0}^{\infty}(c_i^{'}+c_i^{''})\int_{-\infty}^{\infty} x^r H_i(x)\Phi(x)\phi(x)\mathrm{d}x \\ &\quad -\sum_{i=0}^{\infty}\sum_{j=1}^{\infty}(c_i^{''}c_j^{'}+c_i^{'}c_j^{''})\int_{-\infty}^{\infty} x^r H_i(x)H_{j-1}(x)\phi^2(x)\mathrm{d}x \\ &= \sum_{i=0}^{\infty}(c_i^{'}+c_i^{''})l_{ir}^{'} - \sum_{i=0}^{\infty}\sum_{j=1}^{\infty}(c_i^{''}c_j^{'}+c_i^{'}c_j^{''})l_{ijr}^{''}. \end{aligned} \tag{D.18}$$

□

In practice $l_{ir}^{'}$ and $l_{ijr}^{''}$ can be computed offline numerically for bounded r.

Table D.1: $\mathsf{E}[Y^r]$ of standard distributions using Gram-Charlier series with $Q = 4$.

Distribution	$r=1$	$r=2$	$r=3$	$r=4$
normal	0.564	1	1.410	3
uniform	0.022	0.083	0.215	1.8
exponential	0.837	4.019	11.502	36.703

Next, we will express the moments of a distribution function corresponding to the nth order statistics using the Gram-Charlier series given by Eq. (D.1). The derivation shows that the moments can be expressed explicitly as shown by the following theorem.

Theorem D.2 *The rth moment of Y of Gram-Charlier series is given by*

$$\begin{aligned}\mathsf{E}[Y^r] = \lim_{p\to\infty} n\sum_{i=0}^{\infty}\sum_{m=0}^{n-1} c_i\binom{n-1}{m} \\ \sum_{\substack{r_1,r_2,\ldots,r_p\geq 0 \\ r_1+r_2+\cdots+r_p=n-m-1}} \binom{n-m-1}{r_1\, r_2\, \ldots r_p}(-1)^{n-m-1}\prod_{j=1}^{p}(c_j)^{r_j} l_{irmnp}\end{aligned} \tag{D.19}$$

where l_{irmnp} will be computed offline numerically and it is given by

$$l_{irmnp} = \int_{-\infty}^{\infty} x^r H_i(x)\Phi^m(x)\phi^{n-m}(x)\prod_{j=1}^{p}(H_{j-1}(x))^{r_j}\mathrm{d}x. \tag{D.20}$$

Proof
From Eq. (D.17) providing that the moments of X exist, then the moments of Y are given by

$$\mathsf{E}[Y^r] = \int_{-\infty}^{\infty} y^r f_Y(y)\mathrm{d}y$$

$$= n \int_{-\infty}^{\infty} x^r f_X(x)(F_X(x))^{n-1}\mathrm{d}x$$
$$= n \int_{-\infty}^{\infty} x^r (\sum_{i=0}^{\infty} c_i H_i(x)\phi(x))(\Phi(x) - \sum_{j=1}^{\infty} c_j H_{j-1}(x)\phi(x))^{n-1}\mathrm{d}x. \quad \text{(D.21)}$$

Using the binomial probability law

$$(a+b)^n = \sum_{i=0}^{n} \binom{n}{i} a^i b^{n-i} \quad \text{(D.22)}$$

we obtain

$$\begin{aligned}
\mathsf{E}[Y^r] &= n \int_{-\infty}^{\infty} x^r \sum_{i=0}^{\infty} c_i H_i(x)\phi(x) \sum_{m=0}^{n-1} \binom{n-1}{m} \Phi^m(x)(-\sum_{j=1}^{\infty} c_j H_{j-1}(x)\phi(x))^{n-m-1}\mathrm{d}x \\
&= n \sum_{i=0}^{\infty} \sum_{m=0}^{n-1} \binom{n-1}{m} c_i \int_{-\infty}^{\infty} x^r H_i(x)\Phi^m(x)\phi(x)(-\phi(x)\sum_{j=1}^{\infty} c_j H_{j-1}(x))^{n-m-1}\mathrm{d}x \\
&= n \sum_{i=0}^{\infty} \sum_{m=0}^{n-1} \binom{n-1}{m} (-1)^{n-m-1} c_i \\
&\quad \int_{-\infty}^{\infty} x^r H_i(x)\Phi^m(x)\phi^{n-m}(x)(\sum_{j=1}^{\infty} c_j H_{j-1}(x))^{n-m-1}\mathrm{d}x.
\end{aligned}$$

When we place c_j outside the integration and apply the multinomial probability law, we obtain Eq. (D.19).
□

Unlike Eq. (D.13), Eq. (D.19) is much more complicated. Hence, we conclude that the use of the Gram-Charlier series is prohibited for the analysis of parallel programs.

Appendix E

Source Code

In this appendix, we present an instrumented version of a Quicksort program. This program shows how counters are placed to generate memory references to an input array.

```
/*-------------------------------------------------------------------------
 *      generate memory reference to array arr[]
 *      the Quicksort algorithm
 *      instrumented version
 *-------------------------------------------------------------------------
 */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "qscache.h"
#include "../sorting/nrutil.c"
#include "../seq/rvgen.c"
#define SWAP(a,b) temp=(a);(a)=(b);(b)=temp;
#define M 7
#define NSTACK 500
/* Here M is the size of subarrays sorted by straight insertion and
NSTACK is the required auxiliary storage. */

#define c_num           27
float   exec_time;
int     c_if[c_num],counter[c_num];

void sort(unsigned long n, float arr[])
/* sorts an array arr[1...n] into ascending numerical order using the
quicksort algorithm. n is input; arr is replaced on output by its
sorted rearrangement.
*/
{
        unsigned long i,ir=n,j,k,l=1;
        int jstack = 0, *istack;
        float a,temp;

        istack=ivector(1,NSTACK);
        for(;;) {       /* Insertion sort when subarray small enough. */
                if(ir-l < M) {
                        for(j=l+1;j<=ir;j++) {
                                c_if[0]++;
                                if(hit(j-1,&exec_time))
                                        counter[0]++;
                                a=arr[j];
                                for(i=j-1;i>=1;i--) {
                                        c_if[1]++;
                                        if(hit(i-1,&exec_time))
                                                counter[1]++;
                                        if(arr[i] <= a) break;
                                        c_if[2]++;
```

```
                        if(hit(i-1,&exec_time))
                                counter[2]++;
                        c_if[3]++;
                        if(hit(i,&exec_time))
                                counter[3]++;
                        arr[i+1]=arr[i];
                }
                c_if[4]++;
                if(hit(i,&exec_time))
                        counter[4]++;
                arr[i+1]=a;
        }
        if(jstack == 0) break;
        ir=istack[jstack--]; /* pop stack and begin */
        l=istack[jstack--];  /* a new round of partitioning */
} else {        /* choose median of left, center, and right */
        k=(l+ir) >> 1; /* elements as partitioning element a. */
        c_if[5]++;
        if(hit(k-1,&exec_time))
                counter[5]++;
        c_if[6]++;
        if(hit(l,&exec_time))
                counter[6]++;
        SWAP(arr[k],arr[l+1]); /* Also rearrange so that */
        c_if[7]++;
        if(hit(l,&exec_time))
                counter[7]++;
        c_if[8]++;
        if(hit(ir-1,&exec_time))
                counter[8]++;
        if(arr[l+1] > arr[ir]) { /* a[l+1]<=a[l]<=a[ir]. */
                c_if[9]++;
                if(hit(l,&exec_time))
                        counter[9]++;
                c_if[10]++;
                if(hit(ir-1,&exec_time))
                        counter[10]++;
                SWAP(arr[l+1],arr[ir]);
        }
        c_if[11]++;
        if(hit(l-1,&exec_time))
                counter[11]++;
        c_if[12]++;
        if(hit(ir-1,&exec_time))
                counter[12]++;
        if(arr[l] > arr[ir]) {
               c_if[13]++;
                if(hit(l-1,&exec_time))
                        counter[13]++;
                c_if[14]++;
                if(hit(ir-1,&exec_time))
                        counter[14]++;
                SWAP(arr[l],arr[ir]);
        }
        c_if[15]++;
        if(hit(l,&exec_time))
                counter[15]++;
        c_if[16]++;
        if(hit(l-1,&exec_time))
                counter[16]++;
        if(arr[l+1] > arr[l]) {
                c_if[17]++;
                if(hit(l,&exec_time))
                        counter[17]++;
                c_if[18]++;
                if(hit(l-1,&exec_time))
                        counter[18]++;
                SWAP(arr[l+1],arr[l]);
        }
```

```
                        i=l+1;          /* Initialize pointers for partitioning */
                        j=ir;
                        c_if[19]++;
                        if(hit(l-1,&exec_time))
                                counter[19]++;
                        a=arr[l];       /* Partitioning element */
                        for(;;) {       /* Beginning of innermost loop. */
                                do {
                                        i++;
                                        c_if[20]++;
                                        if(hit(i-1,&exec_time))
                                                counter[20]++;
                                } while(arr[i] < a);
                                        /* Scan up to find element > a */
                                do {
                                        j--;
                                        c_if[21]++;
                                        if(hit(j-1,&exec_time))
                                                counter[21]++;
                                } while(arr[j] > a);
                                        /* Scan down to find element < a */
                                if(j < i) break;
                                /* Pointers crossed. Partitioning complete. */
                                c_if[22]++;
                                if(hit(i-1,&exec_time))
                                        counter[22]++;
                                c_if[23]++;
                                if(hit(j-1,&exec_time))
                                        counter[23]++;
                                SWAP(arr[i],arr[j]);
                                        /* Exchange element */
                        }       /* End of innermost loop */
                        c_if[24]++;
                        if(hit(l-1,&exec_time))
                                counter[24]++;
                        c_if[25]++;
                        if(hit(j-1,&exec_time))
                                counter[25]++;
                        arr[l]=arr[j];
                        c_if[26]++;
                        if(hit(j-1,&exec_time))
                                counter[26]++;
                        arr[j]=a;
                        jstack += 2;
                        /* Push pointers to larger subarray on stack, */
                        /* process smaller subarray immediately. */
                        if(jstack > NSTACK)
                                nrerror("NSTACK too small in sort.\n");
                        if(ir-i+1 >= j-l) {
                                istack[jstack]=ir;
                                istack[jstack-1]=i;
                                ir=j-1;
                        } else {
                               istack[jstack]=j-1;
                                istack[jstack-1]=l;
                                l=i;
                        }
                }
        }
        free_ivector(istack,1,NSTACK);

}
```

Appendix F

Glossary of Symbols and Abbreviations

The following symbols and abbreviations are sometimes used:

ARP	Alternating Renewal Process
$B(a, b)$	the beta function with parameter a and b
cdf	cummulative distribution function
ε_r	the relative error of the rth raw moment
$\mathrm{E}(\theta)$	exponential distribution with parameter θ
E	mean
FCFS	First Come First Served
GBD	generalized beta distribution
GLD	generalized lambda distribution
iid	identical and independent distributed
Kur	kurtosis
$\mathrm{N}(\mu, \sigma)$	normal distribution with mean μ and standard deviation σ
NAS-EP	Numerical Aerospace Simulation-Embarrassingly Parallel
P	probability
PSRS	Parallel Sorting by Regular Sampling
pdf	probability density function
pmf	probability mass function
RTL	Run Time Library
Skw	skewness
SSS	Straight Selection Sort
$\mathrm{U}(a, b)$	uniform distribution with sample space $[a, b]$
Var	variance
VS	Vector Scaling

Bibliography

[1] V.S. Adve and M.K. Vernon, "The influence of random delays on parallel execution times," in *Proceedings of the 1993 ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems*, May 1993, pp. 61–73.

[2] V.S. Adve and M.K. Vernon, "A Deterministic Model for Parallel Program Performance Evaluation," Report TR98-333, Rice University, Mar. 1998.

[3] F. Allen, M. Burke, R. Cytron, J. Ferrante, W. Hsieh and V. Sarkar, "A framework for determining useful parallelism," in *Proc. 1988 Int'l Conf. Parallel Proc.*, IEEE, Aug. 1988, pp. 207–215.

[4] I. Angus, G. Fox *et al.*, *Solving problems on concurrent processors, Software for concurrent processors*, vol. II. Englewood Cliffs, NJ: Prentice Hall, 1990.

[5] D. Atapattu and D. Gannon, "Building analytical models into an interactive prediction tool," in *Proc. Supercomputing '89*, ACM, 1989, pp. 521–530.

[6] T.S. Axelrod, "Effects of synchronization barriers on multiprocessor performance," *Parallel Computing*, vol. 3, May 1986, pp. 129–140.

[7] J. Baeten and P. Weijland, *Process Algebra*. Cambridge University Press, 1990.

[8] D. Bailey, E. Barszcz and J. Barton *et al.*, "The NAS Parallel Benchmarks," Report RNR-94-007, Department of Mathematics and Computer Science, Emory University, Mar. 1994.

[9] V. Balasundaram, G. Fox, K. Kennedy and U. Kremer, "A static performance estimator to guide data partioning decisions," in *Proc. 3rd ACM SIGPLAN Symposium on PPoPP*, Apr. 1991, pp. 213–223.

[10] T. Ball and J.R. Larus, "Branch prediction for free," *ACM SIGPLAN Notices*, vol. 28, June 1993, pp. 300–313.

[11] T. Ball and J.R. Larus, "Optimally profiling and tracing programs," *ACM Transactions on Programming Languages and Systems*, vol. 16, July 1994, pp. 1319–1360.

[12] Y. Bard, "Some extensions to multiclass queueing network analysis," in *Performance of Computer Systems* (A. Butrimenko M. Arato and E. Gelenbe, eds.), North-Holland, 1979.

[13] F. Baskett, K.M. Chandy, R.R. Muntz and F.G. Palacios, "Open, closed, and mixed networks of queues with different classes of customers," *Journal of the ACM*, vol. 22, Apr. 1975, pp. 248–260.

[14] B. Beizer, *Micro-Analysis of Computer System Performance*. New York: Van Nostrand Reinhold Company, 1978.

[15] J.A. Bergstra, *Handbook of Process Algebra*. Amsterdam: Elsevier Science Ltd., Mar. 2001.

[16] K.O. Bowman and L.R. Shenton, "Distribution of the ratio of gamma variates," *Journal of Statistical Computation and Simulation*, vol. 27, no. 1, 1998, pp. 1–19.

[17] B. Calder, D. Grunwald, D. Lindsay, J. Martin, M. Mozer and B.G. Zorn, "Corpus-based static branch prediction," in *Proceedings of the ACM SIGPLAN'95 Conf. on Programming Language Design and Implementation*, La Jolla, California, 18–21 June 1995, pp. 79–92.

[18] M.J. Clement and M.J. Quinn, "Multivariate statistical techniques for parallel performance prediction," in *Proc. 28th Hawaii Int'l Conf. on System Sciences, Vol. II*, IEEE, Jan. 1995, pp. 446–455.

[19] B. Dodin, "Bounding the project completion time distributions networks," *Operations Research*, vol. 33, no. 4, 1985, pp. 862–881.

[20] C. Donnelly and R. Stallman, *Bison: The YACC-Compatible Parser Generator*. Boston: Free Software Foundation, 2002.

[21] E.R. Dougherty, *Probability and Statistics for Engineering, Computing, and Physical Sciences*. New Jersey: Prentice-Hall, Inc., 1990.

[22] E.J. Dudewicz and S.N. Mishra, *Modern Mathematical Statistics*. New York: John Wiley & Sons, Inc., 1988.

[23] M. Evers, S.J. Patel, R.S. Chappell and Y.N. Patt, "An analysis of correlation and predictability: What makes two-level branch predictors work," in *The 25th Annual Int'l Symposium on Computer Architecture*, 1999, pp. 52–61.

[24] T. Fahringer and H.P. Zima, "A static parameter-based performance prediction tool for parallel programs," in *Proc. 7th ACM Int'l Conf. on Supercomputing*, Tokyo, July 1993, pp. 207–219.

[25] J.A. Fisher and S.M. Freudenberger, "Predicting conditional branch directions from previous runs of a program," *ACM SIGPLAN Notices*, vol. 27, Sept. 1992, pp. 85–95.

[26] M. Freimer, G. Mudholkar, G. Kollia and C.T. Lin, "A study of the generalized tukey lambda family," *Communications in Statistics, Theory and Methods*, vol. 17, Oct. 1988, pp. 3547–3567.

[27] E. Gelenbe, E. Montagne, R. Suros and C.M. Woodside, "Performance of block-structured parallel programs," in *Parallel Algorithms and Architectures* (M. Cosnard *et al.*, eds.), Amsterdam: North-Holland, 1986, pp. 127–138.

[28] A.J.C. van Gemund, "Performance prediction of parallel processing systems: The PAMELA methodology," in *Proc. 7th ACM Int'l Conf. on Supercomputing*, Tokyo, July 1993, pp. 318–327.

[29] A.J.C. van Gemund, "Compile-time performance prediction of parallel systems," in *Proc. Computer Performance Evaluation: Modelling Techniques and Tools (Tools'95), LNCS 977*, Heidelberg, Sept. 1995, pp. 299–313.

[30] A.J.C. van Gemund, "Performance modeling of parallel systems," PhD thesis, Delft University Press, Apr. 1996.

[31] A.J.C. van Gemund, "The importance of synchronization structure in parallel program optimization," in *Proc. of the 11th Int'l Conf. on Supercomputing*, ACM Press, 1997, pp. 164–171.

[32] A.J.C. van Gemund, "Symbolic performance performance modeling of parallel systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 14, Feb. 2003, pp. 154–165.

[33] A. Gonzalez-Escribano, "Synchronization architecture in parallel programming models," Ph.D. thesis, Departemento de Informática Universidad de Valladolid, July 2003.

[34] A. Gonzalez-Escribano, A.J.C. van Gemund and V. Cardenoso-Payo, "Mapping unstructured applications into nested parallelism," in *Proc. Int'l Conf. on Vector and Parallel Processing (Vecpar'02)*, Porto, Portugal, June 2002, pp. 469–482.

[35] N. Götz, U. Herzog and M. Rettelbach, "Multiprocessor and distributed system design: The integration of functional specification and performance analysis using stochastic process algebra," in *Performance Evaluation of Computer and Communication Systems* (L. Donatiello and R. Nelson, eds.), Springer, 1993.

[36] E.J. Gumbel, "Statistical theory of extreme values (main results)," in *Contributions to Order Statistics* (A.E. Sarhan and B.G. Greenberg, eds.), New York: John Wiley & Sons, 1962, pp. 56–93.

[37] F. Hartleb and V. Mertsiotakis, "Bounds for the mean runtime of parallel programs," in *Proc. 6th Int'l Conf. Modelling Techniques and Tools for Comp. Perf. Eval.*, Edinburgh, Sept. 1992, pp. 197–210.

[38] P. Heidelberger and K.S. Trivedi, "Analytic queueing models for programs with internal concurrency," *IEEE Transaction on Computers*, vol. 32, Jan. 1983, pp. 73–82.

[39] J.L. Hennessy and D.A. Patterson, *Computer Architecture (2nd ed.): A Quantitative Approach*. Morgan Kaufmann Publishers Inc., 1996.

[40] J. Hillston, "*A Compositional Approach to Performance Modelling*," PhD. thesis, University of Edinburgh, 1994.

[41] C.A.R. Hoare, "Communicating sequential processes," *Communications of the ACM*, vol. 21, Aug. 1978, pp. 666–677.

[42] C.A.R. Hoare, *Communicating Sequential Processes*. Prentice-Hall, 1985.

[43] K.E. Iverson, *A Programming Language*. Wiley, 1962.

[44] N.C. Matals J.A. Geenwood, J.M. Landwehr and J.R. Wallis, "Probability weighted moments: Definition and relation to parameters of several distribution expressible in inverse form," *Water Resource Research*, vol. 15, 1979.

[45] K. Jensen, "Coloured Petri nets: A high-level language for system design and analysis," in *High level Petri Nets: Theory and Application* (K. Jensen and G. Rozenberg, eds.), Springer Verlag, 1991, pp. 44–122.

[46] M.A. Johnson and M.R. Taffe, "Matching moments to phase distributions: Mixtures of erlang distribution of common order," *Stochastic Models*, vol. 5, 1989, pp. 711–743.

[47] H. Jonkers, "Performance analysis of parallel systems: A hybrid approach," Ph.D. thesis, Delft University of Technology, Oct. 1995.

[48] A. Kapelnikov, R.R. Muntz and M.D. Ercegovac, "A modeling methodology for the analysis of concurrent systems and computations," *Journal of Parallel and Distributed Computing*, vol. 6, 1989, pp. 568–597.

[49] Z.A. Karian and E.J. Dudewicz, "Fitting the generalized lambda distribution to data: A method based on percentiles," *Communications in Statistics - Simulation and Computation*, vol. 28, no. 3, 1999, pp. 793–819.

[50] Z.A. Karian, E.J. Dudewicz and P. McDonald, "The extended generalized lambda distribution system for fitting distributions to data: History, completion of theory, tables, applications, the final word on moment fits," *Communications in Statistics - Simulation and Computation*, vol. 25, 1996, pp. 611–642.

[51] Khuri and I. André, *Advanced Calculus with Applications in Statistics*. New York: Wiley, 1993.

[52] R.A.R. King and H.L. Mac Gillivray, "A starship fitting method for the generalized lambda distribution," *Journal of Statistics*, vol. 41, no. 3, 1999, pp. 353–374.

[53] W. Kreutzer, *System simulation, programming styles and languages*. Addison-Wesley, 1986.

[54] C.P. Kruskal and A. Weiss, "Allocating independent subtasks on parallel processors," *IEEE Transactions on Software Engineering*, vol. 11, Oct. 1985, pp. 1001–1016.

[55] A. Lakhany and H. Mausser, "Estimating the parameters of the generalized lambda distributions," *Algo Research Quarterly*, vol. 3, Dec. 2000, pp. 47–58.

[56] S.S. Lavenberg, *Computer Performance Modeling Handbook*. New York: Academic Press, 1983.

[57] E.D. Lazowska *et al.*, *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*. Englewood Cliffs, N.J.: Prentice-Hall, 1984.

[58] A. Leon-Garcia, *Probability and Random Processes for Electrical Engineering*. New York: Addison-Wesley Publishing Company, Inc., 1994.

[59] B.P. Lester, "A system for the speedup of parallel programs," in *Proceedings of the 1986 Int'l Conf. on Parallel Processing*, IEEE, Aug. 1986, pp. 145–152. Maharishi Int'l U, Fairfield, Iowa.

[60] K. Li, "Stochastic bounds for parallel program execution times with processor constraints," *IEEE Transactions on Computers*, vol. 46, no. 5, 1997, pp. 630–636.

[61] D.-R. Liang and S.K. Tripathi, "On performance prediction of parallel computations with precedent constraints," *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, no. 5, 2000, pp. 491–508.

[62] J. Lüthi, S. Majumdar, G. Kotsis and G. Haring, "Performance bounds for distributed systems with workload variabilities and uncertainties," *Parallel Computing*, vol. 22, Feb. 1997, pp. 1789–1806.

[63] S. Madala and J.B. Sinclair, "Performance of synchronous parallel algorithms with regular structures," *IEEE Transactions on Parallel and Distributed Systems*, vol. 2, Jan. 1991, pp. 105–116.

[64] V.W. Mak and S.F. Lundstrom, "Predicting performance of parallel computations," *IEEE Transactions on Parallel and Distributed Systems*, vol. 1, July 1990, pp. 257–270.

[65] Waterloo Maple Web Site, `http://www.waterloomaple.com/`.

[66] C.L. Mendes, J.-C. Wang and D.A. Reed, "Automatic performance prediction and scalability analysis for data parallel programs," in *Proc. Second Workshop on Automatic Data Layout and Performance Prediction*, Houston, Apr. 1995.

[67] J.F. Meyer, A. Movaghar and W.H. Sanders, "Stochastic activity networks: Structure, behavior, and application," in *Proc. of the Int'l Conf. on Timed Petri Nets*, Torino, July 1985, pp. 106 – 115.

[68] R.E. Mortensen, "Alternating renewal process models for electric power system loads," *IEEE Transactions on Automatic Control*, vol. 35, Nov. 1990, pp. 1245–1249.

[69] T. Murata, "Petri nets: Properties, analysis and applications," *Proceedings of the IEEE*, vol. 77, Apr. 1989, pp. 541–580.

[70] E.J. Muth, "A method for predicting system downtime," *IEEE Transactions on Reliability*, vol. R-17, June 1968, pp. 97–102.

[71] M.F. Neuts, *Matrix-geometric Solutions in Stochastic Models*. Baltimore, MD.: ohns Hopkins University Press, 1981.

[72] D.M. Olsson and L.S. Nelson, "Nelder-Mead simplex procedure for function minimization," *Technometrics*, vol. 17, 1975, pp. 45–51.

[73] A. Ozturk and R.F. Dale, "Least squares estimation of the parameters of the generalized lambda distribution," *Technometrics*, vol. 27, Feb. 1985, pp. 81–84.

[74] PAMELA Project Web Site, `http://rama.pds.twi.tudelft.nl/~hasyim/Pamela`.

[75] S-T. Pan, K. So and J.T. Rahmeh, "Improving the accuracy of dynamic branch prediction using branch correlation," *ACM SIGPLAN Notices*, vol. 27, Sept. 1992, pp. 76–84.

[76] V. Paxson, *Flex, version 2.5*. Berkeley: University of California, 1995.

[77] J.L. Peterson, *Petri Net Theory and Modelling of Systems*. Englewoods Cliffs, NJ,: Prentice-Hall Inc., 1981.

[78] C.A. Petri, "Communication with Automata," Tech. Rep. RADC-TR-68-305, Griffiss Air Force Base, 1966.

[79] T. Pham-Gia and N. Turkkan, "System availability for a gamma alternating renewal process," *Naval Research Logistics*, vol. 46, 1999, pp. 822–844.

[80] C.D. Polychronopoulos, M. Girkar, M.R. Haghighat, C.L. Lee, B. Leung and D. Schouten, "Parafrase-2: An environment for parallelizing, partitioning, synchronizing, and scheduling programs on multiprocessors," in *Proc. 1989 Int'l Conf. Parallel Proc.*, IEEE, Aug. 1989, pp. II:39–48.

[81] W.H. Press, S.A. Teukolsky, W.T. Vetterling and B.P. Flannery, *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge: Cambridge University Press, second ed., 1992.

[82] M.J. Quinn, *Parallel Computing: Theory and Practice*. New York: McGraw-Hill, second ed., 1994.

[83] J.S. Ramberg, P.R. Tadikamalla, E.J. Dudewicz and F.M. Mykytka, "A probability distribution and its uses in fitting data," *Technometrics*, vol. 21, May 1979, pp. 201–214.

[84] G.L. Reijns, A.J.C. van Gemund and H. Gautama, "On the use of Pierson distributions for the performance prediction of parallel programs," in *Symposium on Performance Evaluation: Stories and Perspectives*, OCG, Dec. 2003, pp. 365–380.

[85] M. Reiser and S.S. Lavenberg, "Mean value analysis of closed multichain queueing networks," *Journal of the ACM*, vol. 27, Apr. 1980, pp. 313–322.

[86] W. Reisig, *Petri Nets*. Springer Verlag, 1985.

[87] J.T. Robinson, "Some analysis techniques for asynchronous multiprocessor algorithms," *IEEE Transactions on Software Engineering*, vol. 5, Jan. 1979, pp. 24–31.

[88] S.M. Ross, *Stochastic Processes*. New York, USA: John Wiley & Sons, Inc., second ed., 1996.

[89] R.A. Sahner and K.S. Trivedi, "Performance and reliability analysis using directed acyclic graphs," *IEEE Transactions on Software Engineering*, vol. 13, Oct. 1987, pp. 1105–1114.

[90] W.H. Sanders, W.D. Obal, M.A. Qureshi and F.K. Widjanarko, "The UltraSan modeling environment," *Performance Evaluation Journal, special issue on Performance Modeling Tools*, 1995.

[91] V. Sarkar, "Determining average program execution times and their variance," in *Proceedings of the SIGPLAN '89 Conf. on Programming Language Design and Implementation*, 1989, pp. 298–312.

[92] V. Sarkar, *Partitioning and Scheduling Parallel Programs for Multiprocessors*. London: Pitman, 1989.

[93] J.M. Schopf, "A practical methodology for defining histograms for predictions and scheduling," in *Parallel Computing: Fundamentals & Applications, Proceedings of the Int'l Conf. ParCo'99, 17-20 August 1999, Delft, The Netherlands* (E.H. D'Hollander, J.R. Joubert, F.J. Peters and H. Sips, eds.), Imperial College Press, Apr. 2000, pp. 664–671.

[94] J.M. Schopf and F. Berman, "Performance prediction in production environments," in *Proceedings of the 1st Merged Int'l Parallel Processing Symposium and Symposium on Parallel and Distributed Processing (IPPS/SPDP-98)*, Los Alamitos, IEEE Computer Society, Mar. 30–Apr. 3 1998, pp. 647–653.

[95] J.M. Schopf and F. Berman, "Using stochastic information to predict application behavior on contended resources," *Int'l Journal of Foundations of Computer Science*, vol. 12, no. 3, 2001, pp. 341–364.

[96] P. Schweitzer, "Approximate analysis of multiclass closed networks of queues," in *Proc. of Int'l Conf. on Control and Optimization*, Amsterdam, 1979.

[97] H. Shi and J. Schaeffer, "Parallel sorting by regular sampling," *Journal of Parallel and Distributed Computing*, vol. 14, no. 4, 1992, pp. 361–372.

[98] A.W. Shogan, "Bounding distributions for a stochastic pert network," *Networks*, vol. 7, 1977, pp. 359–381.

[99] A.J. Smith, "Cache memories," *ACM Computing Surveys*, vol. 18, no. 3, 1982, pp. 473–530.

[100] A.J. Smith, "Cache evaluation and the impact of workload choice," in *Proc. of the 12'th Int'l Symp. in Comp. Arch.*, Boston, Massachusset, June 1985, pp. 64–75.

[101] J.E. Smith, "A study of branch prediction strategies," in *Proceedings of 8th Annual Int'l Symposium on Computer Architecture*, May 1981, pp. 135–148.

[102] K. So, A.S. Bolmarcich, F. Darema and V.A. Norton, "A speedup analyzer for parallel programs," in *Proc. 1987 Int'l Conf. Parallel Proc.*, IEEE, Aug. 1987, pp. 653–661.

[103] F. Sötz, "A method for performance prediction of parallel programs," in *Proc. CONPAR 90-VAPP IV (LNCS 457)* (H. Burkhart, ed.), Springer-Verlag, 1990, pp. 98–107.

[104] A. Stuart and J.K. Ord, *Kendall's Advanced Theory of Statistics*, vol. 1. New York: Halsted Press, 6th ed., 1994.

[105] A. Thomasian and P. Bay, "Analytic queuing network models for parallel processing of task systems," *IEEE Transactions on Computers*, vol. 35, Dec. 1986, pp. 1045–1054.

[106] K.S. Trivedi, *Probability and Statistics with Reliability, Queuing and Computer Science Applications*. Englewood Cliffs, New Jersey: Prentice-Hall, 1982.

[107] C. van Reeuwijk, "Tm: A code generator for recursive data structures," *Software: Practice and Experience*, vol. 22, Oct. 1992, pp. 899–908.

[108] C. van Reeuwijk, A.J.C. van Gemund and H.J. Sips, "Spar: A programming language for semi-automatic compilation of parallel programs," *Concurrency: Practice and Experience*, vol. 9, no. 11, 1997, pp. 1193–1205.

[109] T.A. Wagner, V. Maverick, S.L. Graham and M.A. Harrison, "Accurate static estimators for program optimization," *SIGPLAN Notices*, vol. 29, June 1994, pp. 85–96.

[110] K-Y. Wang, "A framework for static, precise performance prediction for superscalar-based parallel computers," in *Proc. 4th Int'l Workshop on Compilers for Parallel Computers*, Delft, The Netherlands, Dec. 1993, pp. 413–427.

[111] D.A. Wood and .D. Hill, "Cost-effective parallel computing," *IEEE Computer*, vol. 28, no. 2, 1995, pp. 69–72.

[112] D.A. Wood, M.D. Hill and R.E. Kessler, "A model for estimating trace-sample miss ratios," *1991 ACM SIGMETRICS Int'l Conf. on Measurement and Modeling of Computer Systems ACM SIGMETRICS Performance Evaluation Review*, vol. 19, no. 1, May 21-24, 1991.

[113] Y. Wu and J.R. Larus, "Static branch frequency and program profile analysis," in *Int'l Symposium on Microarchitecture (MICRO-27)*, 1994, pp. 1–11.

[114] N. Yazici-Pekergin and J-M. Vincent, "Stochastic bounds on execution times of parallel programs," *IEEE Transactions on Software Engineering*, vol. 17, no. 10, Oct, 1991, pp. 1005–1012.

[115] T.Y. Yeh and Y.N. Patt, "Alternative implementations of two-level adaptive branch prediction," in *19th Int'l Symposium on Computer Architecture*, Gold Coast, Australia, ACM and IEEE Computer Society, 1992, pp. 124–134.

[116] J. Zahorjan, "Balanced job bound analysis of queuing networks," *Communication of the ACM*, vol. 25, Feb. 1982, pp. 134–141.

Samenvatting

Prestatiemodellering speelt een belangrijke rol in het voorspellen van de gevolgen van een ontwerpbeslissing en in het diagnostiseren van een specifiek prestatiegedrag. Met name voor complexe systemen zoals parallelle computersystemen kan een gewenst prestatiegedrag bijna niet worden verkregen zonder de hulp van voorspellende modellen.

In de prestatievoorspelling van parallelle programma's onderscheiden we statische en dynamische benaderingen die een geheel andere keuze maken in de fundamentele afweging tussen de hoeveelheid voorspellende informatie en diens betrouwbaarheid. Statische technieken hebben het voordeel dat analytische informatie wordt verkregen over de prestatieeffecten van symbolische programma/machine-parameters, zonder dat dure executie of simulatie is vereist voor elke denkbare input data set. Hun beperkingen in het modelleren van dynamisch gedrag van programma's heeft echter een negatief effect op hun voorspellende nauwkeurigheid. Behalve de toewijzing van gemeenschappelijke middelen is de afhankelijkheid van de invoer een belangrijke oorzaak van dynamisch programmagedrag. Bij veel programma's kan de executietijd enorm varieren, afhankelijk van de invoer, zelfs als de probleemgrootte constant wordt gehouden.

In dit proefschrift wordt een nieuwe benadering introduceerd binnen de symbolische prestatiemodellering van parallelle programma's waarbij de executietijd wordt voorspeld in de vorm van een distributie over een verzameling van mogelijke invoer. De aanpak is gebaseerd op het gebruik van statistische momenten om de distributie te karakteriseren. Algoritmen worden gepresenteerd die een zeer lage tijd- en ruimte-complexiteit hebben die de statistische momenten van de programma-executietijd voorspellen in termen van de momenten van programma-onderdelen die betrokken zijn in de sequentiele, conditionele, en parallelle composities die tot het parallelle programma hebben geleid. De nieuwheid van deze analyse is de algemene toepasbaarheid wegens het gebruik van momenten in tegenstelling tot veel-gebruikte specifieke distribuities, gecombineerd met de zeer lage rekencomplexiteit. De voorspellende nauwkeurigheid van de aanpak is experimenteel getoetst aan de hand van synthetisch verkregen werklastdistributies alsmede distributies verkregen uit bestaande parallelle programma's. Bezien vanuit de zeer lage rekencomplexiteit vormt deze benadering een aantrekkelijke kosten/baten afweging in de analytische prestatiemodellering van parallelle programma's.

Curriculum Vitae

Hasyim Gautama was born in Jember, Indonesia on August 31, 1974. After he completed his high-school education at SMA 1 Jember in 1992, he received the Science and Technology for Industrial Development (STAID) scholarship from the Indonesian government for study in the Netherlands. He first took a Dutch course in the Erasmus Huis, Jakarta, and thereafter he moved to the Netherlands on April 1993. Since then, he joined the Electrical Engineering Department of the Delft University of Technology. On August 1998 he got his Master of Science degree with "cum laude" from the Computer Engineering group under the supervision of Dr.ir. A.J.C. van Gemund. On December 1998 he enrolled as a PhD student under the same supervisor in the same research group. In the middle of his PhD studies he moved to the Software Technology group.

In 1997 he had an internship program in the Dutch TNO research organization in Delft for investigating the performance of T800 Transputer. Between 1999 and 2001 he was involved in the European JOSES-ESPRIT project. His research interest is in the area of performance modeling of parallel systems.

Currently, he works at the CACTUS project in the Delft University of Technology. His research topic in this project is node cooperation in mobile ad hoc networks. After completing this project, he will work for the Indonesian Central Bureau of Statistics (BPS) in Jakarta.

Publications

1. H. Gautama and A.J.C. van Gemund, "Symbolic performance prediction of data-dependent parallel programs", in *IEEE Transactions on Parallel and Distributed Systems*, revision submitted.

2. H. Gautama and A.J.C. van Gemund, "On the use of statistical branch models in static program performance prediction," in *Proc. 11th Intl. Workshop on Compilers for Parallel Computers (CPC'04)*, Seeon, Germany, July 2004, pp. 37–48.

3. H. Gautama and A.J.C. van Gemund, "Symbolic performance estimation of speculative parallel programs," in *Parallel Processing Letters*, vol. 13(4), Dec. 2003, pp. 513–524.

4. H. Gautama and A.J.C. van Gemund, "Symbolic performance prediction of speculative parallel programs," in *Proc. of the 9th Intl. Conference on Parallel Processing (EuroPar 2003)*, Klagenfurt, Austria, Aug. 2003, pp. 88–97. (Distinguished Paper Award)

5. H. Gautama and A.J.C. van Gemund, "Reliability analysis of k-out-of-n systems using generalized lambda distributions," in *Proc. of Intl. Conference on Mathematics and its Applications (SEAM-GMU 2003)*, Yogyakarta, Indonesia, Jul. 2003, pp. 466–474.

6. H. Gautama and A.J.C. van Gemund, "A statistical approach to branch modeling in static program performance prediction", *IPDPS Workshop on Modeling, Evaluation, and Optimization of Parallel and Distributed Systems*, Nice, France, Apr. 2003.

7. H. Gautama and A.J.C. van Gemund, "On the use of Kolmogorov-Smirnov test in performance prediction of fork-join parallel programs," in *Proc. of 7th Indonesian Student's Scientific Meeting*, Berlin, Germany, Oct. 2002, pp. 477–480.

8. H. Gautama and A.J.C. van Gemund, "Toward performance estimation of data-dependent task parallel composition", in *United Kingdom Performance Evaluation Workshop (UKPEW 2002)*, Glasgow, United Kingdom, July 2002, pp. 81-92.

9. H. Gautama and A.J.C. van Gemund, "On the performance estimation of data-dependent task parallel compositions," in *Proc. of 8th Conf. of the Advanced School for Computing and Imaging*, Lochem, The Netherlands, June 2002, pp. 44–50.

10. H. Gautama and A.J.C. van Gemund, "Symbolic performance prediction of data-dependent parallel programs," in *Computer Performance Evaluation, Modeling Techniques and Tools (TOOLS 2002)*, London, United Kingdom, Apr. 2002, pp. 259–278.

11. H. Gautama and A.J.C. van Gemund, "Low-cost performance prediction of data-dependent data parallel programs," in *Proc. of the 9th Intl. Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2001)*, IEEE Computer Society Press, Cincinnati, Ohio, Aug. 2001, pp. 173–182.

12. H. Gautama and A.J.C. van Gemund, "Performance prediction of data-dependent task parallel programs," in *Proc. of the 7th Intl. Conference on Parallel Processing (EuroPar 2001)*, Manchester, United Kingdom, Aug. 2001, pp. 106–116.

13. H. Gautama and A.J.C. van Gemund, "A statistical approach to static performance prediction of branch behavior" in *Proc. of 6th Indonesian Student's Scientific Meeting*, Manchester, United Kingdom, Aug. 2001, pp. 375–378.

14. H. Gautama and A.J.C. van Gemund, "Static cost estimation of data-dependent parallel programs," in *Proc. 11th Intl. Workshop on Compilers for Parallel Computers (CPC'01)*, Edinburgh, United Kingdom, June 2001, pp. 345–356.

15. H. Gautama and A.J.C. van Gemund, "Trade-offs in symbolic cost estimation of parallel programs", in *Proc. of European Joint Conf. on Theory, and Practice of Software (ETAPS 2001)*, Genova, Italy, Apr. 2001, pp. 1–10.

16. H. Gautama and A.J.C. van Gemund, "Static performance prediction of data-dependent programs", *ACM Proc. on The Second Intl. Workshop on Software and Performance (WOSP 2000)*, Ottawa, Canada, Sep. 2000, pp. 216–226.

17. H. Gautama and A.J.C. van Gemund, "Performance prediction of parallel programs based on the use of lambda distributions," in *Proc. of 5th Indonesian Student's Scientific Meeting*, Paris, France, Aug. 2000, pp. 313–316.

18. H. Gautama and A.J.C. van Gemund, "On the use of lambda distributions in parallel program performance prediction", in *Proc. of 6th Conf. of the Advanced School for Computing and Imaging*, Lommel, Belgium, June 2000, pp. 343–349.

19. H. Gautama and A.J.C. van Gemund, "A statistical approach to program performance prediction," in *Proc. of 4th Indonesian Student's Scientific Meeting*, Kassel, Germany, Oct. 1999, pp. 91–94.

20. H. Gautama and A.J.C. van Gemund, "A probabilistic approach to embedded program performance prediction," in *Proc. of 5th Conf. of the Advanced School for Computing and Imaging*, Heijen, The Netherlands, June 1999, pp. 65–72.