Chapter 4
Introduction to Optimization Under Uncertainty Techniques for High-Performance Multicore Embedded Systems Compilation

Oana Stan and Renaud Sirdey

Abstract The compilation process design for massively parallel multicore-embedded architectures requires solving a number of difficult optimization problems, nowadays solved mainly using deterministic approaches. However, one of the main characteristics of these systems is the presence of uncertain data, such as the execution times of the tasks. The authors consider that the embedded systems design is one of the major domains for which applying optimization under uncertainty is legitimate and highly beneficial. This chapter introduces the most suitable techniques from the field of optimization under uncertainty for the design of compilation chains and for the resolution of the associated optimization problems.

4.1 Introduction

At the beginning of the twenty-first century, it has become obvious that the performances of single core architectures reached a plateau, the main reasons being the limits of instruction-level parallelism (ILP) as well as the heat wall for the frequency [39].

Between the only viable solutions left to improve performance was to make use of additional high-level parallelism, i.e., multiply the number of processing elements per chip. As a consequence, the standard mainstream and embedded architectures include, nowadays, at least four generic cores and we are entering into a multicore era in which the updated Moore's law states that *the number of cores doubles every two years*.

Figure 4.1 [53] captures the general exponential evolution trend of the number of individual computing units according to the release years of chips (heterogeneous

O. Stan (✉) · R. Sirdey
Embedded Real Time Systems Laboratory, CEA, LIST,
Point Courrier 172, 91191 Gif-Sur-Yvette, France
e-mail: oana.stan@cea.fr

R. Sirdey
e-mail: renaud.sirdey@cea.fr

M. Fakhfakh et al. (eds.), *Computational Intelligence in Digital and Network Designs and Applications*, DOI 10.1007/978-3-319-20071-2_4

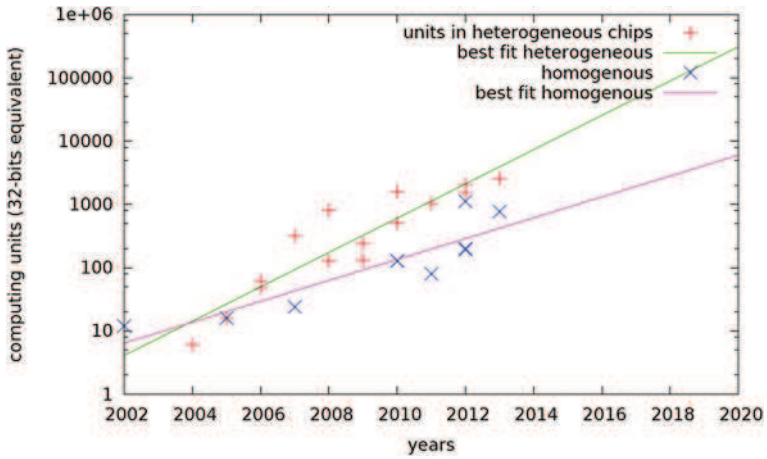

Fig. 4.1 Number of individual processing units in heterogeneous chips (e.g., AMD, NVidia graphics, IBM Cell BE, etc.) and homogeneous chips (e.g., Intel Xeon, IMB Power, STM, Tilera, Kalray, etc.) [53]

and homogeneous). If the generalization of multicore continues according to this trend, at the end of the decade, we will reach at least a thousand of generic cores.

However, according to Gustafson's law [37], as more computing power is available, new and more powerful applications make their appearance in order to benefit from the available capabilities. Already, the latest embedded applications for video and image processing (using complex compression and decompression algorithms), video games, scientific computing, or data security demand a computer power ten to hundred times superior to that of a few years ago.

Therefore, careful attention has to be paid when designing embedded systems solutions since programming applications that fully exploit the computing power and the parallelism is a difficult task.

As we will see further, the design for efficiently embedded manycore systems requires new programming and execution paradigms as well as innovative compilation technologies. One of the common practices is to make use of operation research techniques for the different optimization steps during the compilation process of parallel applications. Since between the main characteristics of the related optimization problems is the presence of intrinsic uncertain parameters, we believe that the overall compilation chain should integrate the latest advances from the optimization field such as stochastic programming methods and robust optimization algorithms.

Thus, this chapter is dedicated to the study of uncertainties associated with the embedded domain and to the analyzis of the most appropriate techniques for optimizing under these uncertainties when designing compilers for parallel embedded applications. The remainder of this chapter is organized as follows: after a short description of manycore architectures in Sect. 4.2.1, we present the existing approaches for programming parallel applications with a particular emphasis on dataflow-based

languages (Sect. 4.2.2). The context and motivation for our study are getting refined in Sect. 4.2.3. Afterward, in Sect. 4.3, we describe the main sources of uncertainty affecting the properties of an embedded environment with a particular focus on execution times. In function of this analysis and taking into account the current state of art from stochastic and robust optimization fields, we give more details in Sect. 4.4 about some of the most relevant models and optimization techniques for the compilation of embedded systems. Also, in the last section, we show how these resolution methods can be applied in an operational manner for concrete application case studies, such as the partitioning, placement, and routing of network processes or the dimensioning of communication buffers.

4.2 Massively Parallel Embedded Systems

According to Flynn's macroscopic classification of computing systems, realized in function of the possible interaction patterns between instructions and data, there are four different theoretical classes of machines: Single Instruction Single Data (SISD), Single Instruction Multiple Data (SIMD), Multiple Instruction Single Data (MISD), and Multiple Instruction Multiple Data (MIMD). Between these categories, only the last three make parallel execution possible, and thus almost all parallel systems today are either SIMDs, easier to program but for which the parallelism is more difficult to exploit, or MIMDs, for which each processor is executing its own program flow. More flexible than SIMD and allowing nonstructured data and conditional or data-dependent algorithms, the MIMD is a more usual implementation of the multi and manycore concept.

Also, according to the memory organization, we can distinguish between DMM (Distributed Memory Machines) and SMM (Shared Memory Machines) systems.

The massively multicore (manycore) type of architecture is a compromise between the DMM and the SMM solving the problem of scalability of the SMMs for which it is difficult to exceed 32 processors and the performance issues of the DMMs.

Let us now give an overview of the main components of a massively multicore architecture, taking as an example the MPPA chip [30].

4.2.1 Manycore Architectures: MPPA Example

A massively multicore (manycore) processor is a parallel computing system, composed of a number of processing cores (at least a dozen), a mix of local and shared memory, distributed global memory or multilevel cache hierarchy, and an infrastructure for intercores communication.

Some examples of such embedded multicore architectures already available nowadays are [3, 10, 30, 42].

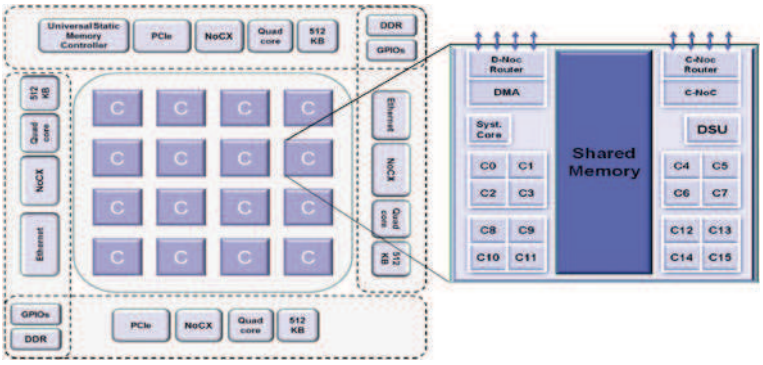

Fig. 4.2 Overview of Kalray's MPPA architecture [30]

The Kalray's MPPA-256 is one of the first homogeneous embedded manycore, released in 2013 and manufactured using 28 nm CMOS technology. As shown in Fig. 4.2, this single-chip manycore processor is organized as 16 (4×4) computing clusters and 4 additionally I/O clusters situated at the periphery and providing access to PCi interfaces, DRAM memory, etc. As the basic processing unit of the MPPA chip, each computing cluster integrates 16 processing engines (PE) cores, one resource management (RM) core, a shared memory, and a direct memory access (DMA) engine for transferring data. The 16 computing clusters as well as the 4 I/O subsystems are connected through a bidirectional NoC (Network-on-Chip) with a 2D torus topology and a wormhole route encoding.

4.2.2 Programming for Manycores: Dataflow Oriented Languages

In order to take benefit of the underlying execution infrastructure, when programming parallel applications for manycore, one has to handle several difficulties: dispose of limited and dependent resources (memory, NoC), be able to run correctly large parallel programs and efficiently exploit the parallelism and the computing power.

There is a real urge nowadays for languages permitting to design efficiently and without difficulties parallel applications. As noticed several years ago, the usual imperative programming paradigms (C or Java like) are based on a sequential von Neumann architecture and thus they are inappropriate for writing effective parallel programs. Other modern programming languages like MPI, OpenMP and OpenCL used currently on distributed systems require explicitly managing communications and synchronizations between tasks.

Some paradigms such as agent-based and dataflow programming languages allow to overcome several of the above drawbacks, providing mechanisms to mask

low-level communication and intertasks synchronization, assure execution determinism and easily integrate existing code.

In a dataflow model, a program is described as a directed graph, consisting of nodes representing tasks (also named agents or actors) and arcs representing unidirectional communications channels. The exchange of data, quantized into tokens, is realized exclusively through the communications channels and the execution of the program consists in a sequence of firings or evaluations (which correspond to the consumption/production of a certain number of tokens).

With the first models emerging in the early 1970s, there are different formalisms for dataflow-based languages (KPN—Kahn Process Networks [44], DPN—Data Process Networks [47], CSDF—Cyclo-Static Data Flows [15], etc.) different in their expressive power and the guarantees they provide.

In the following, we will focus on the CSDF model of computation and on ΣC [2], a more recent dataflow-based language. In a CSDF model, the number of produced/consumed tokens can vary from an activation to another in a cyclic manner. Since programmers do not have to worry about data synchronization and the computing tasks are only connected through well identified channels, this flexible dataflow model is well suited for efficiently scheduling and mapping applications on manycore platforms. ΣC, a C extension programming language based on CSDF, has been designed for allowing reusability of existing C code, while taking advantage of the properties of a dataflow models such as the ability to verify absence of deadlocks and memory bounded execution. Its ability to specify the production and consumption of each task is used at compile time for different checkings such as buffer sizing, placement, routing, parallelism refinement, etc.

Thus, once the application has been designed and implemented using a parallel programming language, it is the role of the compilation chain to make the connection with the specific execution model for the embedded manycore target. A general difference between a dataflow compiler and a standard one is the fact that the former is handling itself the underlying parallelism, easing the role of the designer.

Let us exemplify the compilation process through the ΣC compilation chain.

4.2.3 Compilation of Applications for Manycore Systems: ΣC Toolchain Example

The ΣC compilation toolchain adapts as best as possible the application code, generic, to the targeted architecture: number of cores, NoC topology, etc. As such, even if the language is platform independent, the compiler will automatically map the parallel program onto a large number of processors, using different architectures and communication types.

There are four passes in which ΣC compilation process is organized:

- **Lexical analysis, parsing and code generation**. This first pass, the ΣC front-end, begins with a lexical, syntactic, and semantic analysis of the code, common to most

compilers. Afterward, preliminary C codes are generated from ΣC sources either for offline execution (the instantiation codes of the agents) or for further refinement.

- **Compilation of the parallelism**. The purpose of the second pass, the ΣC middle-end, is to instantiate and connect the agents, by executing at compile time the corresponding codes generated by the first pass.
  Once the construction of the application graph is complete, parallelism reduction techniques by pattern matching [21] are applied and a safe computation of a deadlock-free lowest bound for the buffers sizes of the links is also performed.
- **Resource allocation**. The third pass is in charge of resource allocation (in the larger sense). First, it supports a dimensioning of communication buffers taking into account the execution times of the tasks and the application requirements in terms of bandwidths (nonfunctional constraints). Next, in order to realize a connection with the execution model, it constructs a folded unbounded partial ordering of task occurrences (and thus, finitely representable).
  This pass is also responsible of placement and routing, with the objectives of grouping together (under capacity constraints for each cluster of the architecture) the tasks which communicate the most, mapping these groups of tasks to the clusters and finally, computing routing paths for the data traversing the NoC.
- **Runtime generation and link edition**. The last pass, the ΣC back-end, is responsible of generating the final C code and the runtime tables. Also, during this stage and using C back-end compiler tools, the link edition, and the loadbuild are realized.

4.2.4 Characteristics of Optimization Problems Associated to the Compilation Process for Manycore

As seen in the previous section, the compilation process of an application for a massively parallel architecture is becoming rather complex and requires solving several difficult optimization problems.

Nowadays, the compiler design implies the application of advanced operations research techniques not only to the so-called back-end (by solving optimization problems such as buffer sizing and instruction scheduling, e.g., [14, 38, 49]) but also more upward and all the long of the compilation process, in order to efficiently allocate and exploit the interrelated resources offered by parallel architectures. Between the more recent optimization problems, we can mention the placement/routing for multicores or the construction of a partial order under throughput constraints (e.g., [35, 36]).

Moreover, most of the existing studies treating optimization problems for embedded parallel architectures propose deterministic models and we noticed only a few studies which take into consideration parameter variations and apply the techniques of optimization under uncertainty to the embedded domain (e.g., [22, 48, 51]).

Still, one of characteristics of the manycore systems is the presence of intrinsic uncertain data occurring in the definition of these related optimization problems,

such as execution times or latencies. As we will see further, experimental studies from both fields of operations research and program compilation have shown that considering a fixed value for the uncertain parameters, respectively, execution times (usually the mean value), can lead to wrong estimates and optimization solutions not always feasible. As such, developing and testing optimization techniques taking into account uncertainty for this field seem beneficial and even necessary.

In order to conceive and develop methods of optimization under uncertainty which are adequate to the domain of compilation for manycores, a first step consists in identifying, analyzing, and, if possible, modeling the sources of uncertainty specific to this area. As such, we proceed in the next section with a qualitative analysis of the uncertainty sources, with a particular emphasis on the execution times.

4.3 Characterization of the Uncertainties in the Context of Manycores

One of the main sources of uncertainties related to the domain of embedded systems lies in the intrinsic indeterminism of execution times for computing kernels of intermediate granularity.

4.3.1 Overview of the Execution Times

In fact, there are two main sources of uncertainty related to the execution times of embedded systems:

1. intrinsic dependency on the data. Since usually the computation code of an application depends on the input data, there are several treatments which could be executed by the application, translating into different data-dependent paths, with potentially different execution times.
2. extrinsic uncertainty due to architecture characteristics. Variations of execution times are also related to the speculative components (such as caches, pipelines, or branch prediction) of modern hardware architectures on which the application is executed.

These sources of uncertainty are not independent and one must take into account both execution paths and hardware mechanisms.

As described previously, we assume that the embedded application consists of a number of tasks or agents, which work together to achieve the required functionalities. In Fig. 4.3 [71] several relevant properties of the execution time for a task are revealed. The darker upper curve represents the set of all execution times. The shortest execution time is often called best-case execution time (BCET) and the longest is called worst-case execution time (WCET). The other envelope represents a subset of the measures of the execution times. The minimum and maximum of the lower curve

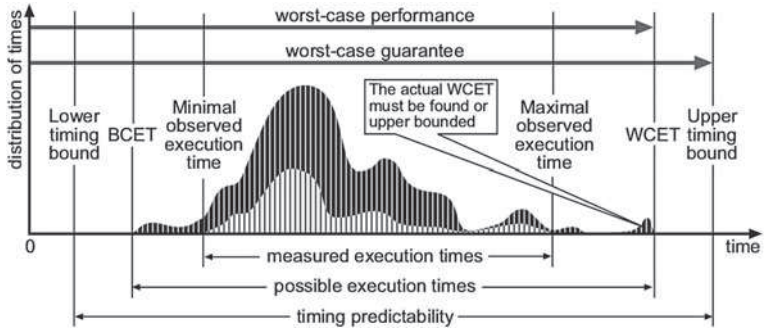

Fig. 4.3 Some properties of the execution times of a real-time task [71]

are the minimal and the maximal observed execution times, respectively. Since, in most cases, the space of all possible executions is too large to fully explore, and also because of the undecidability problem associated to the running of an arbitrary program, it is not possible to determine the exact worst and best case execution times.

Most researches dedicated to the timing analysis of execution times consist in deriving or computing upper bounds for the WCET. For a detailed overview and survey of methods and tools for estimating WCET, we refer the reader to [71].

4.3.2 Estimating Execution Times Distributions

While the methods for estimating bounds for execution times are getting more and more complex, by also taking into account the speculative behavior of the target architecture, they remain justified mainly for hard real-time systems. Instead, for soft real-time systems, there are more and more studies based on probabilistic analysis and approaches for scheduling (e.g., [17, 29, 55]) considering that the execution times of the tasks follow probability distributions.

The problem of estimating the execution times consists in predicting the execution time of a task on a variety of machines in function of the data set and with a high level of accuracy. The existing solutions to this problem can be divided into three main classes: code analysis [63], analytic profiling [33, 45, 73], and statistic prediction [28, 43].

An execution time estimate found by analysis of the source code of a task is typically limited to a specific class of architectures and a particular code type. Consequently, code analysis is not very adapted to treat heterogeneous computing. The profiling technique, first presented by Freund [33], determines the composition of a task in terms of primitive code types. Code profiling data is then combined with benchmark data (obtained on each machine and measuring the performance for each code type). The main disadvantage of this type of methods is that they cannot determine the variations in the input data set. The third category, the statistical prediction

algorithms, makes predictions from previous observations. Each time a task executes on a machine, the execution time is measured and added to the set of past observations. The quality of estimation depends on the size of the set of observations. The advantage is that these methods can compensate for parameters of the input data and the machine type without any supplementary information about the internal code or the machine.

A recent work [56] is going further with the analysis, by studying the variations of execution times on multicore architectures. The experimental work is conducted on samples from two benchmarks; SPEC CPU, large sequential applications, and SPEC OMP2001 benchmarks, OpenMP applications. Each program is executed 30 times on a 8 cores Linux machine with the same training input data each time. The normality check (using the standard Shapiro-Wilk test) for both benchmarks proved that the distribution of execution times is not a Gaussian function in almost all cases. Also, contrary to sequential SPEC CPU applications, OpenMP applications have a more important variability of execution times. By executing 30 times, several applications from the SPEC OMP benchmark on different software configurations (sequential and multithreads), the study shows that if the sequential and single threaded versions do not have important variations, when using a larger thread level parallelism (more than 1 thread), the overall execution times decrease with a deeper disparity. More, the mean confidence intervals (obtained with Student's test) are not always tight.

4.3.3 Execution Times: A Qualitative Analysis and Basic Examples

Even if it is reasonable to assume, in embedded computing, that the execution times have probability distributions of bounded support (no infinite loops), we have to cope with the fact that these distributions are intrinsically multimodal.

Let us give some simple examples. For example, for the computing kernel in Table 4.1, there are two modes for the executions times, possible with different variances, corresponding to the two sequences of instructions (see Fig. 4.4).

Instead, for the code in Table 4.2 with n taking values between 1 and N, S_1 and S_2 being two linear sequences of instructions, the distribution has $2N$ modes (the figure Fig. 4.5 showing a possible envelope of the distribution for the case when $N = 4$).

Running a more complicated application like X264 encoder clearly shows that the distribution of execution times is difficult to model and that it is multimodal.

Table 4.1 Example 1—A simple code snippet

```
if condition then
   S1
else
   S2
end if
```

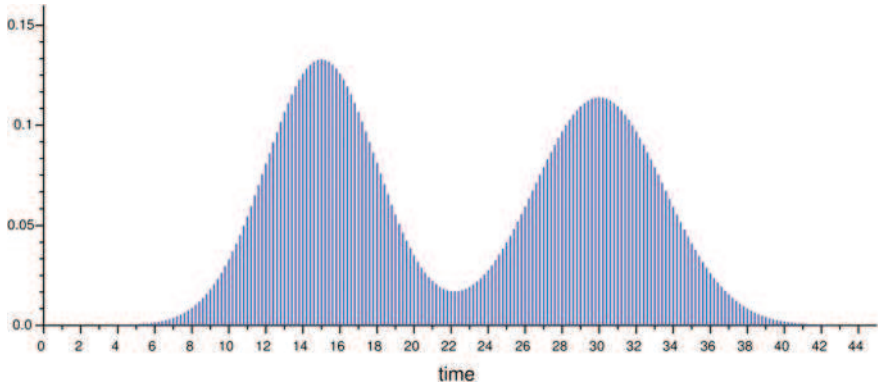

Fig. 4.4 Example 1—2 mode distribution for the execution times

Table 4.2 Example 2—Another code snippet

```
for i = 1 to n do
  if condition then
    S1
  else
    S2
  end if
end for
```

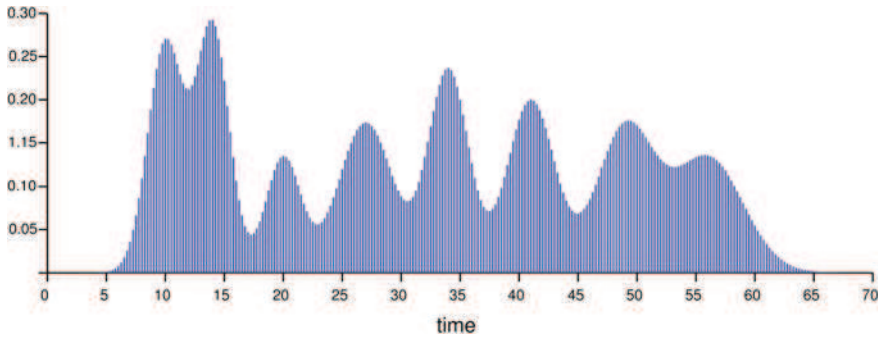

Fig. 4.5 Example 2—Multimodal distribution for the execution times

Figure 4.6 shows the envelope of executions times for each frame when the X264 is executed on a Linux machine, taking as input a video benchmark of size 1280×720, with 24 frames per second.

Hence, it is difficult to model these probabilities laws through usual distributions such as the normal or uniform ones, which are unimodal.

Furthermore, in the case of a process network, we cannot overlook the problem of dependency between these random variables. An easy example consists in the target tracking pipeline for which the execution times of each of the pipeline elementary

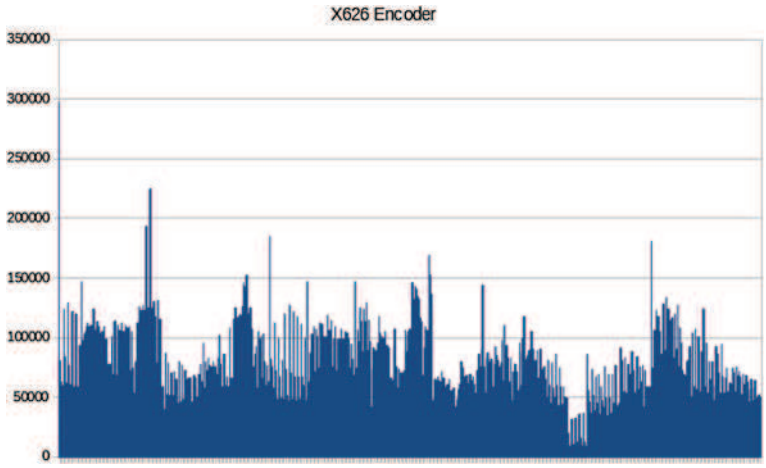

Fig. 4.6 Example 3—Envelope of execution times for frame treatment in a X264 encoder

tasks depend, to a certain degree, on the number of effectively treated targets. In Table 4.3, another example is presented consisting of two elementary tasks both depending on same input data d, difficult to characterized, and each task having two modes for its execution times. As such, as illustrated in Fig. 4.7, the execution time of task $T1$ is dependent to a certain degree of execution time of task $T2$.

To conclude, it is appropriate to assume that the execution times are random variables characterized by complicated multimodal joint distributions, presumably better defined as unions of orthotopes rather than a Gaussian or even a mixture of Gaussians.

However, modeling the underlying distribution of execution times seems delicate and thus, for solving the optimization problems related to compilation for high parallel systems, we do not encourage the use of parametric methods. The main raison is that these parametric optimization methods are making assumptions on the existence of a probability model for the uncertain parameters.

Let us now introduce some general principles about optimizing under uncertainty.

Table 4.3 Exemple 4—Code snippet showing possible tasks dependency

T_1	T_2
if $f(d)$ **then**	**if** $g(d)$ **then**
S_1	S_3
else	**else**
S_2	S_4
end if	**end if**

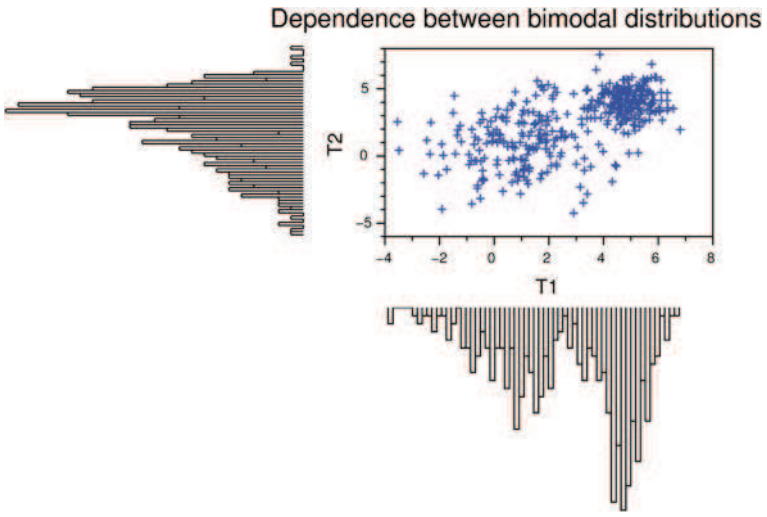

Fig. 4.7 Example 4—Execution times for T_1 and T_2

4.4 Optimization Under Uncertainty

Beginning with the seminal works of Dantzig [25], Charnes and Cooper [23], Miller and Wagner [57], Bellman and Zadeh [4], optimization under uncertainty remains one of the most active domains of research and thanks to recent studies allowing major advances, there is an increased regain of interest for this discipline.

4.4.1 Generalities

An example illustrating the importance of taking into account uncertainty for optimization problems is the recent case study of Ben-Tal and Nemirovski [8] on a collection of 90 problems from NETLIB library [70]. They showed that systems optimized in the classical deterministic sense can be very sensitive to small changes on the parameters values and that only 1 % perturbation of the data can severely affect the feasibility properties of the found solutions.

Therefore, for real-world optimization problems in which data are uncertain and inexact (as, for example, those related to compilation field), in order to find optimal solutions which are feasible in a meaningful sense, one has to deal with the randomness of the variables.

A crucial aspect being the way uncertainty is formalized, we can thus distinguish two major branches of optimization under uncertainty: stochastic programming and robust optimization.

In stochastic combinatorial optimization problems (SCOP), it is assumed that uncertain information can be described by random variables which can be characterized by probability distributions. Static SCOPs are *a priori* optimization problems where the decisions are taken and optimal solutions are found in the presence of randomness, at one single step, *before* the actual realization of the random variables. Dynamic SCOPs consider that decision cannot be made until random variables are revealed and associated random events have happened. As such, decisions are taken *after* random events occur in a single stage, in the case of *simple recourse* problems or in several stages, for *multistage recourse* problems. For both static and dynamic models, there are decisions and there are observations of the random variables, the order of succession being given by different schemes: for static models, first decision, then observation while for dynamic problems, at least one decision is preceded by at least one observation.

Robust optimization does not need to assume any exact knowledge about the probability distribution of random data; instead, uncertain information is set based. As such, uncertain parameters are characterized through a set of possible events and, usually, the decision-making process searches for solutions that are feasible for any realization of the uncertainty in the given set. Indeed, the main criticism of classical robust approaches (e.g., the so-called "max-min" or worst-case approach, the regret maxmin, etc.) is their over-conservatism since they are searching for solutions that are feasible for *all* possible events from the uncertainty set. As such, the obtained solutions are often too conservative, of large cost, being guaranteed even for events with a low probability to occur. Recent approaches (e.g., [7, 13]), more flexible, try to rectify this drawback, by making particular assumptions about the uncertainty set of the parameters and proposing deterministic counterparts to the original robust problem.

Even if the robust methods construct solutions which are immune to data uncertainty, in general, the quality of the solution is not assessed with probabilistic considerations. However, from our perspective, the probability to respect a given reliability target is a more intuitive notion, often easier to set for a decision maker. Additionally, we consider that the problem formulations accompanied by probability guarantees are more appropriate when applying optimizations for the domain we target, the compilation for manycore.

As such, in the following, we are concentrating on the resolution methods for static stochastic programs and (without loss of generality) with uncertainty affecting the constraints, aka *chance-constrained programs*.

4.4.2 Chance-Constrained Programming

The general form of the chance-constrained problem is the following:

$$\begin{aligned} \min_{x} \quad & g\,(x) \\ \text{s.t.} \quad & \mathbb{P}\,(G(x,\xi) \leq 0) \geq 1-\varepsilon \end{aligned} \tag{CCP}$$

where $x \in \mathbb{R}^n$ is the decision variable vector, $\xi \in \Omega \longrightarrow \mathbb{R}^D$ represents a random vector and $g : \mathbb{R}^n \longrightarrow \mathbb{R}$ is the objective function. We suppose that there exists a probability space $(\Omega, \Sigma, \mathbb{P})$, with Ω, the sample space, Σ, the set of events, i.e., subsets of Ω, and $\mathbb{P}$, the probability distribution on Σ. $G : \mathbb{R}^n \times \mathbb{R}^D \longrightarrow \mathbb{R}^m$ is the function for the m constraints, $0 \leq \varepsilon \leq 1$ is a scalar defining a prescribed probability level and $\mathbb{P}(e)$ of an event e is the probability measure on the set Σ.

This type of problem, where all constraints should be satisfied simultaneously with a probability level of at least $1-\varepsilon$, is known in the literature as a *joint chance constrained program*. Another variant of optimization problems with uncertainty affecting the constraints is the *separate chance constrained program* in which different probability levels ε_i can be assigned to different constraints. In separate chance constraints the reliability is required for each individual feasible region while in joint chance constraints the reliability is assured on the whole feasible space. Even if appealing for their more simple structure, the separate chance-constrained programs have the important drawback of not properly characterizing safety requirements [62]. As such, while separate chance constraints could be used in the case when some constraints are more critical than others, joint chance constraints seems a more appropriate choice for guaranteeing an overall reliability target for the system.

As one may expect, chance-constrained optimization problems are inherently difficult to address and although this class of problems have been studied for the last 50 years, there is still a lot of work to be made towards practical resolution methods. There is not a general method of resolution for chance-constrained programs, the choice of the algorithm depending on the way random and decision variables interact.

Basically, the major difficulties associated to joint CCP are related to the convexity of chance constraints and the evaluation of the probabilistic constraints. For optimization problems, the convexity is a structural property allowing to use resolution techniques from convex optimization field and thus, finding a global optimal solution. Or, the distribution function of random variables is not in general completely concave or convex. Worse, even if each constraint is convex, the union of all of them may not be convex. As for the evaluation, for a given x, of the probability that $G(x,\xi) \leq 0$, one has to know the probability distribution of the random vector ξ. So, a first problem raises, the one of the modeling random data in practical applications when the involved distributions are not always known exactly and have to be estimated from historical data. The second problem is numerical since typically ξ is multidimensional and there are no ways to compute exactly and efficiently the corresponding probabilities with high accuracy. (At best, if given, the multivariate distribution of ξ, can be approximated by Monte-Carlo simulations or bounding arguments.)

As such, even for simple cases (e.g., Gaussian distributions for random variables), chance-constrained programs can be very difficult to solve. Table 4.4 shows some of the main theoretical and algorithmic resolution methods proposed for solving joint

Table 4.4 Methods for solving chance-constrained programs

Category	Characteristics	Some references
Convexity studies	Theoretical approaches	[23, 57]
	Particular assumptions on the distribution	[40, 62]
Robust optimization	Relatively simple to apply	[5–7, 13, 18–20, 24, 34, 46, 72]
Approximations and sampling	Compute bounds and approximate solutions	[58, 60, 64]
	Usually computationally demanding	[9, 26, 27, 41, 50, 54, 61]
(Meta) Heuristics	Use of precedent techniques for computing distribution	[1, 11, 12, 52, 69]

CCP. Along with the general hypotheses made for each category (e.g., random data in the right-hand side, normal distribution, etc.) (see column "Characteristics") a list of references is provided (in column "Some references").

Concerning the first category, to the best of our knowledge, existing studies determined convexity conditions only for linear probabilistic constraints with normal distributions in left-hand side or log-concave distributions on the right-hand side.

As for the robust approaches proposing probabilistic guarantees, they also need to make particular assumptions, usually quite mild, and they are applicable for specific classes of problems (e.g., linear programs) for an exact resolution.

Other directions of research consist either in discretization and sampling the distribution or in developing convex approximations. Usually, the proposed approximations find feasible but suboptimal and too conservative solutions to the original problem without any guarantees on their quality. The approximation methods based on sampling are replacing the actual distribution by an empirical distribution estimated by simulation when a direct evaluation of the feasibility of chance constraints is not possible and the probability has no available closed form. However, the use of Monte-Carlo simulations is too computationally demanding when ε is small and the assumptions made are restricting their applicability to particular cases (e.g., in order to generate Monte-Carlo samples, these methods require the full joint distribution).

Finally, there are a few approaches that propose heuristics, type genetic algorithms, or tabu search, combined with simulation techniques, in order to propose approximate solutions to chance-constrained programs.

We will not go further on in details concerning each class of methods and we refer the interested reader to the articles mentioned in Table 4.4. Instead, in the next section, we will concentrate on a selection of resolution approaches for chance-constrained programs, the most appropriate (from our perspective) for the field of compilation for high parallel embedded systems.

4.5 Some Suitable Methods of Optimization Under Uncertainty for Compilation of High Parallel Embedded Systems

One of the key aspects when choosing an optimization algorithm consists in analyzing the specificities of the parameters of the problem and of the involved area. Or, most of the previously mentioned approaches for optimizing under uncertainty are making assumptions (e.g., existing analytical form of the distribution, independence of the random vector components) which are either restrictive, or difficult to verify, or not always adequate to represent the uncertainty of real-life applications. Also, most of the approaches for optimization under uncertainty, based on probability models, make assumptions on the underlying distribution or use simulations without making a true connection with the data (i.e., without a through model validation with the available experimental data samples).

However, as illustrated before, the main sources of uncertainty for the compilation of dataflow application on manycore, the execution times, are random variables difficult to fully describe analytically and for which it is difficult to assume a "nice" probability model.

Another aspect to take into account when conceiving optimization methods for dimensioning embedded applications is their response-time requirements.

For safety-critical applications (hard real-time systems), like nuclear power plant control or flight management systems, all the timing constraints have to be met which often goes along with a worst-case approach. Even if they lead to an oversizing of the systems, worst-case approaches [65] are favored since missing any deadline is highly risky and unacceptable.

The methodologies we present in the next section are more suitable for the dimensioning of *soft real-time systems*, such as multimedia applications (video encoding, virtual reality, etc.) for which missing a deadline from time to time is acceptable, resulting only in a decrease of the quality of service. In fact, almost all of the probability-based studies related to real-time systems are intended for this kind of systems. Thus, even if the dimensioning is no longer guaranteed in all cases, acceptable deviations are admitted, and, in consequence, it is possible to avoid oversizing (expensive in terms of hardware) or undersizing (expensive in terms of reliability). Moreover, for a system already dimensioned, it is possible to estimate the level of robustness and specify deviation scenarios for which the system is no feasible or scenarios which could be acceptable.

As such, in the following, we describe two methodologies for optimization under uncertainty adapted for the compilation of soft real-time applications. Since the choice of a proper probability model for the execution times seems difficult, these methods are nonparametric with almost none or only a few assumptions on the distributions of the uncertain data.

4.5.1 The Robust Binomial Approach

The robust binomial approach (RBA) [68] is a simple and pragmatic method using statistical hypothesis testing theory and directly exploiting an available sample, with almost no assumption on the uncertain data. Since it is a specialization of the sample-based approaches, which have the drawback to have a high computational complexity, this generic method is intended to be used within an heuristic algorithm. Moreover, it leads to a generic solution to leverage existing heuristic algorithms for the deterministic case to their chance-constrained counterparts to solve relatively *large size* optimization problems.

4.5.1.1 Basic Ideas and Motivation

In the framework of an iterative compilation, we have at our disposal representative experimental temporal data, obtained during system validation, for example, when performing tests on the target architecture. These observations can be directly employed in order to construct an equivalent optimization problem, more robust and compatible with the variations of the real data, with the condition that the available sample is sufficiently representative of the entire distribution.[1]

The idea is to take advantage of the existing experimental data and revisit the scenario approach in order to provide probabilistic guarantees. The general scenario approach [19, 20] is between the only tractable convex approximations of a chance-constrained program, which does not impose any restrictions on the structure of the uncertain data (in particular with respect to the random vector component independence). Given a sample $\xi^{(1)}, \ldots, \xi^{(NS)}$ of size NS of independent and identically distributed observations of the random vector ξ, the scenario approach in its original form searches a solution satisfying all the realizations and therefore it finds suboptimal solutions, too conservative.

4.5.1.2 Statistical Hypothesis Testing

Before presenting the statistical results on which the robust binomial approach is based, let us introduce the following notation:

[1] An assumption that can be in practice checked, to some extent, using static program analysis techniques. An assumption which also relies reasonably on the expertise of test engineers in terms of designing validation cases representative of real-world system operations.

x	decision vector
ξ	uncertainty vector
p_0	$\mathbb{P}(G(x,\xi) \leq 0)$
$\xi^{(1)}, \ldots, \xi^{(NS)}$	i.i.d. random variables corresponding to NS observations of ξ
$\tilde{\xi}^{(i)}$	realization of observation $\xi^{(i)}$
χ_i	Bernoulli variable equal to 1 if $G\left(x, \xi^{(i)}\right) \leq 0$ and 0 otherwise.

The random variable $\chi = \sum_{i=1}^{NS} \chi_i$ follows, *by definition*, a Binomial distribution with parameters NS and p_0 ($\chi \sim \mathscr{B}(NS, p_0)$). Let us now consider a realization $\tilde{\chi}$ of χ. If $\tilde{\chi}$ (corresponding to the number of times the inequality $G(x,\xi) \leq 0$ is satisfied on a sample of size NS) is sufficiently large (for instance, larger than $k(NS, 1-\varepsilon, \alpha)$) we say that the constraint $\mathbb{P}(G(x,\xi) \leq 0) \geq 1-\varepsilon$ is *statistically satisfied*.

The value of the threshold $k(NS, 1-\varepsilon, \alpha)$ (to which, for simplicity sake, we will refer, from now on, as k) will be chosen so that the probability we accept the constraint by error is smaller than a fixed α, in which case p_0 is strictly smaller than $1-\varepsilon$:

$$\mathbb{P}(\chi \geq k) \leq \alpha \tag{1}$$

For any fixed $p_0 < 1-\varepsilon$, $\mathbb{P}(\chi \geq k)$ is smaller than $\mathbb{P}(\chi' \geq k)$ when $\chi' \sim \mathscr{B}(NS, 1-\varepsilon)$. So we can choose k such that $\mathbb{P}(\chi' \geq k) \leq \alpha$.

Given x and ε, the parameter α can be interpreted as the type I error of the statistical hypothesis test with the following composite hypothesis:

$$\begin{cases} \mathrm{H}_0 : \mathbb{P}\left(G(x,\xi) \leq 0\right) < 1-\varepsilon \\ \mathrm{H}_1 : \mathbb{P}\left(G(x,\xi) \leq 0\right) \geq 1-\varepsilon \end{cases}$$

H_0 corresponds to the null hypothesis made by caution, which is (intuitively) the hypothesis we wish to reject only if we have statistically significant reasons to do so.

Hence, we can conclude, with a high *confidence level* of at least $1-\alpha$, that $p_0 \geq 1-\varepsilon$.

4.5.1.3 Sensitivity Analysis of the Parameters in RBA

Table 4.5 shows some minimal values for k in function of the sample size NS, of $\varepsilon = 0.10$ and of $\alpha = 0.05$. For example, for establishing that an inequality holds with a preset probability level of $1-\varepsilon = 0.90$ and with a confidence level of $1-\alpha = 0.95$, for a sample of size 50, the threshold k needed is at 48 and $P(\chi \geq 48) \approx 0.034$. It should also be noted that, for a practical use, the parameters ε and α should be of the same order of magnitude.

We can also establish in advance the minimal size of the sample required for a fixed level of the probability $1-\varepsilon$ (with $\varepsilon \in\ ]0, 1[$) and a prespecified confidence level $1-\alpha$ (with $\alpha \in\ ]0, 1[$).

Table 4.5 Examples values for $k(NS, 0.90, 0.05)$ in function of NS

NS	$k(NS, 0.90, 0.05)$
30	29
40	38
50	48
100	94
1000	915
10,000	9528

In particular, if $p_0 = 1 - \varepsilon$ and $\mathbb{P}(\chi = NS) > \alpha$ then we can affirm that the sampling size is insufficient (which is true for $NS = 10$ and $NS = 20$). This formula provides an easy way to determine the minimal number of realizations that need to be drawn in order to statistically significantly (α) achieve the desired probability level ($1 - \varepsilon$).

A further analysis consists in studying the effect of variations of $1 - \alpha$ and of ε on the threshold k. For an acceptable risk error α (less than 10 %) and a fixed probability level $1 - \varepsilon$, the variation of k in function of α does not look so important (in average a difference of 7 additional realizations for respecting the constrains and accept a smaller risk of 0.01 instead of 0.1 for a sample size of 1000). Instead, the value of the initial reliability level $1 - \varepsilon$ has a greater impact on the threshold k for same sample size. As expected, for an important probability guarantee, the number of realizations satisfying the constraints has to be higher. For example, for a sample of size 1000 and different levels of $1 - \alpha$, we remark an augmentation in the number of realizations to hold the constraints (i.e., the value of k) of 85, in average, for $\varepsilon = 0.01$ than for $\varepsilon = 0.1$.

4.5.1.4 Chance Constraints and Sampling

The statistical theory explained above can be applied for obtaining a statistically significant approximation model to the initial program (CCP). In order to obtain a relevant equivalent program to (CCP) model, the following assumptions about the random vector ξ represented by a sample of size NS are made:

Assumption 1 NS, the size of the sample for the uncertain vector ξ, is finite and sufficiently representative.

Assumption 2 The sample for ξ is composed of independent and identically distributed (i.i.d.) observations: $\xi^{(1)}, \ldots, \xi^{(NS)}$.

The first assumption is not very restrictive, since even if the number of initial observations is not sufficient, we can resort to statistical methods for resampling, such as bootstrapping [31]. However, it is important that the initial sample is representative of the distribution. The second assumption, of independence, is on the different observations of the random vector and not on its components which (as already stated) can

be dependent. Additionally, the assumptions above remain quite general. As many previous studies do not mention, these assumptions are also necessary in the case of methods using a probability model, as the model itself must be validated e.g., on a Kolmogorov–Smirnov hypothesis test *using an i.i.d. sample of experimental data*.

Let us now define the binary variable $\tilde{\chi}_i$ for realization $\tilde{\xi}^i$:

$$\tilde{\chi}_i = \begin{cases} 1 & \text{if } G\left(x, \tilde{\xi}^{(i)}\right) \leq 0, \\ 0 & \text{otherwise.} \end{cases}$$

Since the sum $\sum_{i=1}^{NS} \chi_i$ follows a Binomial distribution of parameters NS and p_0 (again, by construction), it is possible to determine $k(NS, 1-\varepsilon, \alpha)$. Therefore, $\tilde{\chi}_i$, the realization of the variables χ_i, can be used to replace the probability constraint

$$\mathbb{P}(G(x, \xi) \leq 0) \geq 1 - \varepsilon$$

and to obtain the (RBP) formulation, equivalent to (CCP):

$$\min_x \quad g\,(x) \tag{RBP}$$

$$\text{s.t.} \quad \sum_{i=1}^{NS} \tilde{\chi}_i \geq k(NS, 1-\varepsilon, \alpha) \tag{2}$$

$$G(x, \tilde{\xi}^{(i)}) \leq (1 - \tilde{\chi}_i)L; \qquad i = 1, \ldots, NS$$

$$\tilde{\chi}_i \in \{0, 1\}; \qquad i = 1, \ldots, NS$$

The first constraint assures that the number of constraints which are satisfied for the given sample are superior to the threshold k, fixed in advance in function of NS, ε and α. Constraints of type 2 verify the respect of the initial constraint for each realization i, making the link between x, $\tilde{\xi}^{(i)}$ and $\tilde{\chi}_i$, with L a constant of large size, depending on the problem structure but generally easy to find. For example, for a knapsack constraint $\sum_{i=1}^{m} w_i x_i \leq C$ with $w_i \geq 0$ the weights of the items to be placed, supposed uncertain, x_i binary variables and C the maximal capacity allowed, L is equal to $\sum_{i=1}^{m} w_i$.

Minimizing the objective function $g(x)$ for (RBP) model is equivalent to solving the initial program (CCP) with a confidence level of at least $1 - \alpha$. Additionally, we emphasize once more that the validity of this approximation is independent of any particular assumption on the joint distribution of the random vector ξ, in particular with respect to inter-component dependencies.

Although it is well illustrated on the mathematical formulation (RBP), it should be stressed out that RBA approach is not really appropriate in a mathematical programming setting, since, for example, the reformulation of an original linear problem contains many binary variables and it is more complex to deal with. However, the method can be easily and efficiently adapted to heuristic approaches.

4.5.1.5 Adapting (meha) Heuristics with RBA

When disposing of a sample verifying assumptions 1 and 2, by making use of RBA method, any constructive algorithm relying on an oracle for testing solution admissibility can be turned into an algorithm for the stochastic case. This can be done by modifying the said oracle so as to count the number of constraint violations for the given realizations and take an admissibility decision based on the threshold k.

Table 4.6 shows, as an example, the general structure of a greedy algorithm for the deterministic case as well as its adaptation for the stochastic case. The input is problem specific and consists, for the deterministic case, in giving the structure of the objective g, the constraint function G, the parameter vector ξ as well as the domain of definition for the decision variables. For the chance-constrained version, in which we consider ξ as random, we also specify a sample of size NS for ξ, the probability level ε, and, in order to apply the robust binomial approach, the confidence level α. In both cases, R represents the set of decisions not yet made (or residual), D the set of admissible decisions, $g(S)$ the solution value for solution S, d^* the current optimal decision, and S^* the optimal overall solution, build in a greedy fashion. While there are residual decisions to be made, an oracle is evaluating them for deciding the admissible decisions. Between the admissible decisions, only the one with the greatest improvement on the optimal solution value is kept and the overall solution S^* is updated. If no admissible decision is found by the oracle, the algorithm stops. As seen, the only major difference when considering chance constraints is in establishing the set of admissible solutions, **using a stochastic oracle $\mathcal{O}_s$ instead of the original one** $\mathcal{O}$ (line 4). The deterministic oracle is establishing the admissibility of a residual decision by verifying the respect of the constraints, while the stochastic oracle is applying the robust binomial approach and it verifies if a residual decision is stochastically significant with a confidence level of $1 - \alpha$. For the given sample, it compares the number of constraints respected by the sample with the threshold k, established in advance in function of NS, ε and α (see the procedures for $\mathcal{O}$ and $\mathcal{O}_s$ in Table 4.7).

Of course, any optimization algorithm relying on an oracle to determine whether or not a solution is admissible (e.g., a neighboring method) can be turned into an algorithm solving the stochastic case using the same method.

4.5.2 Bertsimas and Sim-Like Robust Models

Robust optimization (RO) has gained increasing attention in the last years as another more simple framework for immunizing against parametric uncertainties in an optimization problem. With very few assumptions on the underlying uncertainty, robust methods are designed to solve special classes of programs (e.g., linear models, quadratic programs, etc.) in a mathematical programming setting. Moreover, recent models such as the ones of Ben-Tal and Nemirovski [7] and Bertsimas and Sim [13] are also providing probability guarantees.

Table 4.6 General schema for a constructive algorithm

Deterministic	Stochastic

Require: g and G functions, ξ
$R = \{r : \text{residual decisions}\}$
$S^* = \emptyset$
while $R \neq \emptyset$ **do**
  $D = \{r \in R : \mathscr{O}(r) = True\}$
  if $D \neq \emptyset$ **then**
  $d^* = \underset{d \in D}{\text{argmin}}\ g(S^* \cup \{d\})$
  $S^* = S^* \cup \{d^*\}$
  $R = R \setminus \{d^*\}$
  else
  break;
  end if
end while
Ensure: S^*
	Require: g and G functions
	Require: $\tilde{\xi}_1, \ldots, \tilde{\xi}_{NS}, \varepsilon, \alpha$
	$R = \{r : \text{residual decisions}\}$
	$S^* = \emptyset$
	while $R \neq \emptyset$ **do**
	  $D = \{r \in R : \mathscr{O}_s(r) = True\}$
	  if $D \neq \emptyset$ **then**
	  $d^* = \underset{d \in D}{\text{argmin}}\ g(S^* \cup \{d\})$
	  $S^* = S^* \cup \{d^*\}$
	  $R = R \setminus \{d^*\}$
	  else
	  break;
	  end if
	end while
	Ensure: S^*

Table 4.7 Deterministic oracle versus stochastic oracle

Deterministic oracle $\mathscr{O}$	Stochastic oracle $\mathscr{O}_s$

Require: $r \in R, G, \xi$
if $G(r, \xi) < 0$ **then**
  return $True$
end if
return $False$
Ensure: $True, False$
	Require: $r \in R, G, \tilde{\xi}_1, \ldots, \tilde{\xi}_{NS}$
	Require: $k(NS, \varepsilon, \alpha)$
	$nbRespConstr = 0$
	for $i = 1$ **to** NS **do**
	  if $G(r, \tilde{\xi}_i) < 0$ **then**
	  $nbRespConstr++$
	  end if
	  if $nbRespConstr \geq k$ **then**
	  return $True$
	  end if
	end for
	return $False$
	Ensure: $True, False$

4.5.2.1 Basic Ideas and Motivation

There are two important properties of robust optimization methods that make them appealing in practice [24]. First at all, robust linear models are polynomial in size and can be formalized as linear programs or second-order cone programs. In this way, one can use state-of-the art powerful and efficient LP and SOCP solvers (e.g., CPLEX 9.1) in order to solve small and medium-sized linear problems. Second, the robust methods are not making assumptions on the underlying probability distri-

butions for the stochastic parameters, which, as seen before, there are not always available. Instead, they are applicable for cases in which only some modest distribution information is available (e.g., known mean and support). We can however remark that there are situations in which, even if the probability distribution is available, it is easier to solve the RO-based models than the exact probabilistic formulation (robust solutions obtained with several orders of magnitude faster than the exact solution).

The robust approaches are based on uncertainty sets and, in function of the assumptions made on the properties of these sets, there are different formulations, more or less tractable. With the right class of the uncertainty set, RO tractable models have been found for many well-known classes of optimization problems such as linear, quadratic, semidefinite, or even some cases of discrete problems.

The first robust models constructed feasible solutions for any realization of the uncertainty in the given set, and thus, they were too conservative. Meanwhile, the recent robust approaches permitting to choose a level of probabilistic guarantee allow more flexibility for choosing a trade-off between robustness of their solutions and performance. In contrast to the sensitivity analysis, a post-optimization tool, the probabilistic protection levels for the robust solutions are calculated a priori, in function of the structure and of the size of the uncertainty set.

4.5.2.2 Some Popular Robust Models

As expected, in general, the robust counterpart to an arbitrary optimization problem is of increased computational complexity. However, there are special classes of initial problems and types of uncertainty sets for which the robust version can be handled efficiently. In the following, we will present robust models for linear optimization problems and some general results about other different formulations.

Without loss of generality, the robust counterpart of a linear optimization program (LP) can be written as:

$$\begin{aligned} \min_{x} \quad & c^T x \\ \text{s.t.} \quad & Ax \leq b, \quad \forall A \in \mathscr{U} \end{aligned} \tag{RLP}$$

with A constraint data matrix $m \times n$ and $\mathscr{U}$ the uncertainty set.

Uncertain constraints in LP program were first discussed by Soyster [65] which considered the case when uncertainty is "column-wise," i.e., the columns of the matrix A are data known to belong to convex sets $A_j \in \mathscr{K}_j$ and the constraints are of the form: $\sum_{j=1}^{n} A_j x_j \leq b,\ x \geq 0,\ \forall (A_j \in \mathscr{K}_j,\ j = 1, \ldots, n)$. Soyster showed that, under these constraints, the initial problem is equivalent to

$$\begin{aligned} \min_{x} \quad & c^T x \\ \text{s.t.} \quad & \sum_{j=1}^{n} \bar{A}_j x_j \leq b \\ & x \geq 0 \end{aligned}$$

where $\bar{a}_{ij} = \sup_{A_j \in \mathscr{K}_j}(A_{ij})$. This model is too "pessimistic" since corresponds to the case when every entry of the constraint matrix is as large as possible and no violations of the constraints are allowed. As such, it is a worst-case approach more suitable to solve optimization problems in the context of hard real-time systems since it assures the highest protection against data variations.

In fact, the general case consists in a "row-wise" uncertainty, i.e., the one in which the rows of the constraint matrix are known to belong to convex sets: $a_i \in \mathscr{U}_i,\ i = 1, \ldots, m$. In this case, one has to well specify the properties of uncertainty sets in order to obtain formulations which can be efficiently solved.

Ben-Tal and Nemirovski [7] and El-Ghaoui [32] consider ellipsoidal uncertainty sets and, thus, obtain less conservative models. The assumption made by Ben-Tal and Nemirovski [7] is that the uncertainty set is "ellipsoidal," i.e., an intersection of a finite number of many "ellipsoids"—sets corresponding to convex quadratic inequalities. This leads to an optimization problem over a quadratic constraint[2] and a resulting dual of type second-order cone program (SOCP).

As we will see further, this study is between the first one to propose, under some restrictions, probabilistic guarantees for robust linear programs.

Also, under the same assumption of simple ellipsoidal uncertainty, one can obtain robust counterparts in the form of semidefinite optimization problems (SDF) for other classes of initial programs: quadratically constrained quadratic programs and second-order cone programs.

Les us now present another interesting robust model with different assumptions on the uncertainty set.

4.5.2.3 Bertsimas and Sim Robust Formulation

Let consider a particular row i in the constraint matrix A and J_i the set of coefficients a_{ij} in row i that vary. The model of data uncertainty proposed by Bertsimas and Sim [13] suppose that each uncertain coefficient is a symmetric and bounded random variable $\tilde{a}_{ij}$ taking values in $\left[a_{ij} - \hat{a}_{ij}, a_{ij} + \hat{a}_{ij}\right]$.

The novelty of this approach is the introduction of parameter $\Gamma_i \in [0, |J_i|]$, not necessarily integer, allowing, for each row i, to adjust the robustness of the solutions (against the level of conservatism). As such, a feasible solution is obtained for all

[2] A quadratic constraint is a constraint of type: $\alpha_i^T x + \alpha_i \geq \|B_i x + b_i\|,\ \forall i = 1, \ldots, M$ with α_i fixed reals, a_i and b_i fixed vectors, B_i fixed matrices and $\|.\|$ standing for the usual Euclidean norm.

cases in which up to $\lfloor \Gamma_i \rfloor$ are allowed to vary in the intervals $\left[a_{ij} - \hat{a}_{ij}, a_{ij} + \hat{a}_{ij}\right]$ and one coefficient a_{it} changes by $(\Gamma_i - \lfloor \Gamma_i \rfloor)\hat{a}_{it}$.

Under this assumption, for the initial linear program (3), one can obtain the equivalent linear robust formulation (4). This program which can then be solved using standard linear solvers (e.g., CPLEX, COIN-BC) permits to find approximate robust solutions for problems of small and medium size.

$$\begin{aligned} \min_x \quad & c^T x \\ \text{s.t.} \quad & Ax \leq b \\ & 1 \leq x \leq u \end{aligned} \tag{3}$$

$$\begin{aligned} \min_x \quad & c^T x \\ \text{s.t.} \quad & \sum_j a_{ij} x_j + z_i \Gamma_i + \sum_{j \in J_i} p_{ij} \leq b_i \quad \forall i \\ & z_i + p_{ij} \geq \hat{a}_{ij} y_j \quad \forall i, j \in J_i \\ & -y_j \leq x_j \leq y_j \quad \forall j \\ & l_j \leq x_j \leq u_j \quad \forall j \\ & p_{ij} \geq 0 \quad \forall i, j \in J_i \\ & y_j \geq 0 \quad \forall j \\ & z_i \geq 0 \quad \forall i \end{aligned} \tag{4}$$

Of course, when $\Gamma_i = 0$, the constraints are the same as for the nominal problem (no uncertainty taken into account) and, if $\Gamma_i = |J_i|$ for each i, we obtain Soyster's model.

Another interesting feature for the Bertsimas and Sim method is that it can also be applied, under same hypothesis as before, to some discrete optimization problems, proposing a robust mixed integer linear equivalent to an initial mixed integer program.

4.5.2.4 Probability Guarantees

For linear optimization problems, Bertsimas and Sim [13] as well as Ben-Tal and Nemirovski [7] obtain several probability bounds against constraint violations for different uncertainty formulations.

As such, for constraints of type $\sum_j \tilde{a}_{ij} x_j \leq b_i$, Ben-Tal and Nemirovski [7] assume that the uncertain data are of the form $\tilde{a}_{ij} = (1 + \varepsilon \xi_{ij}) a_{ij}$ with a_{ij}, the nominal value for the coefficient, $\varepsilon \geq 0$ the given uncertainty level, $\xi_{ij} = 0$ for $j \notin J_i$ and $\{\xi_{ij}\}$ random variables independent and symmetrically distributed in $[-1, 1]$.

They then show that, for every i, the probability that constraint:

$$\sum_j a_{ij}x_j + \varepsilon\Omega\sqrt{\sum_j a_{ij}^2 x_j^2} \leq b_i + \delta max\,[1, |b_i|]$$

(where $\delta > 0$ is a given feasibility tolerance and $\Omega > 0$ a reliability parameter) is violated is, at most, $exp(-\Omega^2/2)$.

As for Bertsimas and Sim model [13], the same parameter Γ controls the "price of robustness", i.e., the trade-off between the probability of constraint violation and the feasibility of the solution. So, if more than $\lfloor\Gamma_i\rfloor$ coefficients a_{ij}, $j \in J_i$ are varying, the solution remains feasible with a high probability. Several bounds for probability that the constraints are guaranteed are established for the case when the variables $\tilde{a}_{ij}$ are independent and symmetrically distributed random variables on $\left[a_{ij} - \hat{a}_{ij}, a_{ij} + \hat{a}_{ij}\right]$. Let x^* be the optimal solution of formulation (4). Then, a first bound B_1 for the probability that each constraint i is violated is the following:

$$Pr\left(\sum_j \tilde{a}_{ij}x^* \geq b_i\right) \leq exp\left(-\frac{\Gamma_i^2}{2|J_i|}\right).$$

Under same assumption of independence and symmetry for the random variables, a more tight bound B_2 for violation of constraints is established:

$$Pr\left(\sum_j \tilde{a}_{ij}x^* \geq b_i\right) \leq \frac{1}{2^n}\left\{(1-\mu)\binom{n}{\lfloor\nu\rfloor} + \sum_{l=\lfloor\nu\rfloor+1}^{n}\binom{n}{l}\right\}$$

with $n = |J_i|$, $\nu = (\Gamma_i + n)/2$ and $\mu = \nu - \lfloor\nu\rfloor$.

As for the case when the uncertain data is interdependent, Bertsimas and Sim consider a model in which only $|K_i|$ sources affect the data in the row i and each entry a_{ij}, $j \in J_i$ can be modeled as $\tilde{a}_{ij} = a_{ij} + \sum_{k\in K_i}\tilde{\eta}_{ik}g_{kj}$ where η_{ik} are independent and symmetrically distributed random variables in $[-1, 1]$. Using this model, one can also find a robust linear equivalent formulation for which the probability that the solution remains feasible is guaranteed with high probability (i.e., bound B_1 still holds).

Also, since the assumption of distributional symmetry is too limiting in many real-time situations, Chen et al. [24] propose a generalized framework for robust linear optimization with asymmetric distributions. Using some other deviation measures for random variables such as the forward and the backward deviations, they obtain an equivalent which is a second-order cone program, for which the probability of constraint violation is again being guaranteed to a requested level.

Table 4.8 Example for the values of Γ_i in function of $n = |J_i|$ and a probability of constraint violation of less than 1 % [13]

J_i	Γ_i in function of bound B_1	Γ_i in function of bound B_2
5	5	5
10	9.6	8.2
100	30.3	24.3
200	42.9	33.9
2000	135.7	105

4.5.2.5 Sensitivity Analysis on the Model's Parameters

Table 4.8 shows the choice of Γ_i as a function of $n = |J_i|$ so that the probability that a constraint is violated is less than 1 % and bounds B_1 and, respectively, B_2 are being used. It can be easily seen that the second bound dominates bound B_1 which gives unnecessarily higher values for Γ_i. For example, for $|J_i| = 2000$, in order to have a violation probability of less than 1 %, with B_2, we need to use $\Gamma = 105$ which is only $\approx$17 % of the number of uncertain coefficients. When a lower number of uncertain data is available, (e.g., $|J| = 5$), the model is equivalent to Soyster's formulation since a full protection is necessary. As such, the model of Bertsimas and Sim seems a better choice for finding less conservative solutions for problems with constraints containing a large number of uncertain data.

Moreover, as shown in [12], the Bertsimas's robust model seems to be quite robust when handling deviations in the underlying data: for a portofolio selection problem, for perturbations at 5 % level, the solution frontier is relatively unchanged while the solution frontier for Ben-Tal and Nemirovski approach [7] is severely affected.

4.6 Case Studies

4.6.1 The Partitioning, Placement and Routing of Dataflow Networks

As mentioned in Sect. 4.2.3, one important step in the compilation of dataflow applications for massively parallel systems is the resource allocation, with the associated optimization problems of partitioning, placement, and routing.

In the problem of partitioning of networks of processes, the objective is to assign the tasks to a fixed number of processors in order to minimize communications between processors while respecting the capacity of each processor in terms of resources (memory footprint, core occupancy, etc.). The chance-constrained case considered in [66] consists in taking into account the uncertainty coming from the execution times on the resources defining the weights of the tasks (e.g., core occupancy) and, thus, have probabilistic capacity constraints for the sum of resources

affected to a node. Since a multistart constructive algorithm was already available for solving the nominal case, the authors took advantage of the existing implementation and adapt it, using the robust binomial approach to transform the deterministic admissibility oracle into a stochastic one. The accent is put more on the design-for-use methodology and, for the experimental results, on the price of robustness compared with the deterministic version for different thresholds on the involved parameters—ξ, the probability guarantee, α, the confidence level and N, the size of samples. The importance of taking into account the variations on the weights of the tasks is shown: for half of the stochastic instances, the solutions of the corresponding deterministic problems are not feasible.

Another possible application of the robust binomial approach is for the stochastic problem of joint placement and routing [67] the purpose of it being to map dataflow applications on a clusterized parallel architecture by making sure that the capacities of the clusters are respected and that, for the found placement, there exists a routing through the links of the underlying Network-On-Chip, respecting the maximal available bandwidth. If, again, the resources of the tasks depending on the uncertainty times are uncertain, one has to be able to solve the chance-constrained problem with probability capacity constraints for the nodes. The robust binomial approach was applied in the framework of a GRASP (greedy randomized adaptive search procedure) method and extensive computational tests were performed on 1920 synthetic instances as well as on samples from a real application of motion detection. Different configurations for the model parameters were tested, with $\varepsilon, \alpha \in \{0.01, 0.1\}$. Under same configuration, for more than 70 % of instances, the quality of the stochastic solutions is within 5 % from the quality of deterministic ones. As for the computation time, as expected, it depends on the size of the initial problem: in this case, on the number of clusters and of the size of the available sample.

4.6.2 The Dimensioning of Communications Buffers

Another crucial step in the compilation of an application ΣC is the memory dimensioning for the communication buffers. Bodin et al. [16] treats thus one fundamental problem for the compiler to solve in order to achieve performance, the minimization of the buffer sizes under a throughput constraint. Let us explain more in detail the formulation and point out where the uncertainty is ignored.

As said previously, a CSDF application can be seen as a directed graph $G = (T, A)$ with T, the set of actors (tasks) and A, the set of buffers (arcs or channels). Every actor $t \in T$ is decomposed into $\phi(t) \in \mathbb{N}^*$ phases and each k^{th} phase ($\forall k \in \{1, \ldots, \phi(t)\}$), denoted t_k, has a duration $d(t_k) \in \mathbb{R}$. Also, each actor is executed several times: for every phase k, the n^{th} execution of this phase of task $t \in T$ is $\langle t_k, n\rangle$.

For every couple $(k, n) \in \{1, \ldots, \phi(t)\} \times \mathbb{N}^*$, the preceding execution phase of $\langle t_k, n\rangle$ is:

$$Pred\langle t_k, n\rangle = \begin{cases} \langle t_{k-1}, n\rangle & \text{if } k > 1 \\ \langle t_{\phi(t)}, n-1\rangle & \text{if } k = 1 \end{cases}$$

Every arc $a = (t, t') \in A$ has an associated buffer $b(a)$ from actor t to actor t', containing initially $M_0(a) \in \mathbb{N}$ tokens of stored data. $\forall k \in \{1, \ldots, \phi(t)\}, in_a(k) \geq 0$ data are produced at the end of execution of t_k and, similarly, $\forall k' \in \{1, \ldots, \phi(t')\}$, $out_a(k') \geq 0$ data are consumed before $t'_{k'}$ starts its execution.

Let us also define $I_a\langle t_k, n\rangle$ the total number of data produced by t in buffer $b(a)$ at the completion of $\langle t_k, n\rangle$ which can be computed recursively as: $I_a\langle t_k, n\rangle = I_aPred\langle t_k, n\rangle + in_a(k)$. Similarly, $O_a\langle t'_{k'}, n'\rangle$ is the number of data, computed recursively as $O_aPred\langle t'_{k'}, n'\rangle + out_a(k')$, consumed by t' in buffer $b(a)$ at the completion of $\langle t'_{k'}, n'\rangle$. The amount of tokens, respectively, produced and consumed in $b(a)$ during the entire iteration of actors t and t' are noted $i_a = I_a\langle t_{\phi(t)}, 1\rangle$ and $o_a = O_a\langle t_{\phi(t')}, 1\rangle$.

Let also suppose that each buffer $b(a)$ associated with arc $a = (t, t')$ is bounded. This constraint can be modeled using a feedback arc $a' = (t', t) \in Fb(A)$ for each arch $a = (t, t')$ in A. As such, the initial size of the buffer is $b(a) = M_0(a) + M_0(a')$ and, if $\theta(a) = \theta(a')$ is the size of data stored in $b(a)$, the size of the buffer will be exactly $M_0(a)\theta(a) + M_0(a')\theta(a')$. The whole size of G is then $\sum_{a\in\{A\cup Fb(A)\}} \theta(a)M_0(a)$.

If Th^*_G is the minimal required throughout, the problem consists in finding integer values for each $M_0(a')$ such that the throughput is at least Th^*_G and the whole buffer size is minimal.

Using a periodic schedule, it is possible to model this problem as a linear integer program less general and easier to define. Let S be a function defining a valid schedule that, for each (t, k, n) with $t \in T$, $\forall k \in \{1, \ldots, \phi(t)\}$ and $n \in \mathbb{N}^*$, associates a starting time $S\langle t_k, n\rangle \in \mathbb{R}$ for the nth execution of t_k such that no data is read before it is produced (the number of data in each buffer is positive). The throughput of a periodic actor t is $Th^S_t = \frac{1}{\mu^S_t}$, where μ^S_t is the period of t for schedule S and then the throughout of a schedule is equal to $Th^S_G = \frac{1}{\mu^S_t q_t}$ for any actor $t \in T$ with q_t the repetition vector of G. The inverse of this throughout is the period of the periodic schedule S, defined as $\Omega^S_G = \mu^S_t q_t$.

With this periodic schedule, S are defined precedence constraints on the phase executions of tasks t and t'. As such, for two executions $\langle t_k, n\rangle$ and $\langle t'_{k'}, n'\rangle$ with n and n' in $\mathbb{N}^*$, the arc $a = (t, t')$ and the couple $(k, k') \in \{1, \ldots, \phi(t)\} \times \{1, \ldots, \phi(t')\}$ we define the variables:

$$\alpha^{min}_a(k, k') = \lceil max\{0, in_a(k) - out_a(k')\} + O_a\langle t'_{k'}, 1\rangle - I_a\langle t_k, 1\rangle - M_0(a)\rfloor^{gcd_a}$$
$$\alpha^{max}_a(k, k') = \lfloor O_a\langle t'_{k'}, 1\rangle - I_aPred\langle t_k, 1\rangle - M_0(a) - 1\rfloor^{gcd_a}$$

where gcd_a is the greatest common divisor between i_a and o_a for arc a, $\lceil\alpha\rceil^\gamma = \lceil\frac{\alpha}{\gamma}\rceil \times \gamma$ and $\lfloor\alpha\rfloor^\gamma = \lfloor\frac{\alpha}{\gamma}\rfloor \times \gamma$.

Finally, for every arc a, let define the set of couples $\mathscr{Y}_A = \{(k, k') \in \{1, \ldots, \phi(t)\} \times \{1, \ldots, \phi'(t)\}, \alpha^{min}_a(k, k') \leq \alpha^{max}_a(k, k')\}$.

Let also assume that the period Ω^S_G is fixed in advance. Using the above defined variables, the optimization problem considered can be formulated as the integer linear system (5) [16].

$$\min_{a \in Fb(A)} \quad \theta(a) M_0(a) \tag{5}$$

$$\forall a \in A, \forall (k,k') \in \mathscr{Y}(a), \quad S\langle t'_{k'}, 1\rangle - S\langle t_k, 1\rangle \geq d(t_k) + \Omega_G^S \times \frac{\alpha_a^{max}(k,k')}{i_a \times q_t}$$

$$\forall a \in Fb(A), \forall k \in \{1, \ldots, \phi(t)\}, \forall k' \in \{1, \ldots, \phi(t)\},$$

$$S\langle t'_{k'}, 1\rangle - S\langle t_k, 1\rangle \geq d(t_k) + \Omega_G^S \times \frac{f_a(k,k') \times gcd_a}{i_a \times q_t}$$

$$u_a(k,k') = O_a\langle t'_{k'}, 1\rangle - I_a Pred\langle t_k, 1\rangle - M_0(a) - 1$$

$$f_a(k,k') \times gcd_a \geq u_a(k,k') - gcd_a + 1$$

$$f_a(k,k') \in \mathbb{N}, u_a(k,k') \in \mathbb{N}$$

$$\forall a \in Fb(A), \quad M_0(a) \in \mathbb{N}$$

$$\forall t \in T, \forall k \in \{1, \ldots, \phi(t)\}, \quad S\langle t_k, 1\rangle \in \mathbb{R}^+$$

This mixed integer program is then solved for the deterministic case using the commercial solver Gurobi optimizer tool [59].

One can analyze this model and see that the uncertainties associated with the successive durations of the different phases for tasks executions are localized in the coefficients $d(t_k)$ of the first and second sets of constraints. Therefore, after reformulating the program in a convenient way (i.e., a model similar to (3)) and under the hypothesis of a bounded and symmetric support, one could easily apply Bertsimas and Sim approach [13]. We can then control the robustness of solutions with the parameter Γ which size depends on T, the number of actors, $\phi(t)$ the number of phases for each task t and n, the number of executions for a task.

4.7 Conclusion

The multicore and manycore systems are between the future architectures for server acceleration and embedded devices. In order to fully exploit the huge potential of these architectures, one has to be able to write parallel applications correctly and efficiently. Dataflow paradigms seems a good choice for designing embedded applications without worrying about data synchronization and about the underlying parallelism, left in the "hands" of the compiler.

As such, the high parallel embedded architectures need also innovative compilation techniques, making use of advanced operation research methods for better optimizations. Recent advances in optimization under uncertainty approaches make them appealing in an operational manner for domains where data are uncertain, as it is the case of embedded systems where execution times are subject to variations due to the application inputs and to certain modern hardware characteristics.

Since the execution times are random variables difficult to analytically characterize, the most suitable methods of optimization under uncertainty for embedded applications are nonparametric, with few assumptions on the distribution of the para-

meters involved in the optimization model. In this chapter, we have presented methods from two distinct classes of approaches: the robust binomial approach [68], an extension of the sample-based approaches grounded in statistical testing theory, and the model of Bertsimas and Sim [13], from the robust optimization field. The former, more adapted for a setting where the random variables are dependent and where it is hazardous to make any particular assumption on the distribution, provides approximation solutions for medium and large-sized difficult optimization problems. The latter is more adapted for small and medium optimization problems in order to find exact solutions, when one can make minor assumptions on the support and on the variance of the uncertain parameters. We have also illustrated two possible user cases, appearing in the compilation of cyclo-static dataflow applications.

While this work is only an introduction to the optimization under uncertainty techniques, we hope it is a good starting point for those interested in the design of high performing embedded manycore systems using advanced and robust operation research methods.

References

1. Aringhieri, R.: Solving Chance-constrained Programs Combining Tabu Search and Simulation, pp. 30–41. Springer, Berlin (2004)
2. Aubry, P., Beaucamps, P.E., Blanc, F., Bodin, B., Carpov, S., Cudennec, L., David, V., Dore, P., Dubrulle, P., Dupont, B.d.D., Galea, F., Goubier, T., Harrand, M., Jones, S., Lesage, J., Louise, S., Chaisemartin, N., Nguyen, T., Raynaud, H., Sirdey, R.: Extended cyclostatic dataflow program compilation and execution for an integrated manycore processor. In: Proceedings of the First International Workshop on Architecture, Languages, Compilation and Hardware Support for Emerging Manycore Systems (ALCHEMY 2013), Barcelona, Spain, pp. 1624–1633 (2013)
3. Bell, S., Edwards, B., Amann, J., Conlin, R., Joyce, K., Leung, V., MacKay, J., Reif, M., Bao, L., Brown, J., Mattina, M., Miao, C.C., Ramey, C., Wentzlaff, D., Anderson, W., Berger, E., Fairbanks, N., Khan, D., Montenegro, F., Stickney, J., Zook, J.: TILE64 - processor: a 64-core soc with mesh interconnect. In: IEEE International Solid-State Circuits Conference, ISSCC 2008. Digestof Technical Papers, pp. 88–598 (2008)
4. Bellman, R.E., Zadeh, L.A.: Decision-making in a fuzzy environment. Manag. Sci. **17**(4), B-141–B-164 (1970)
5. Ben-Ameur, W., Kerivin, H.: Routing of uncertain traffic demands. Optim. Eng. **6**(3), 283–313 (2005)
6. Ben-Ameur, W., Zotkiewicz, M.: Robust routing and optimal partitioning of a traffic demand polytope. Int. Trans. Oper. Res. **18**(3), 307–333 (2011)
7. Ben-Tal, A., Nemirovski, A.: Robust solutions of linear programming problems contaminated with uncertain data. Math. Program. **88**, 411–424 (2000)
8. Ben-Tal, A., Nemirovski, A.: Robust optimization: methodology and applications. Math. Program. **92**(3), 453–480 (2002)
9. Ben-Tal, A., Nemirovski, A.: On safe tractable approximations of chance-constrained linear matrix inequalities. Math. Oper. Res. **34**, 1–25 (2009)
10. Benini, L., Flamand, E., Fuin, D., Melpignano, D.: P2012: building an ecosystem for a scalable, modular and high-efficiency embedded computing accelerator. In: Design, Automation Test in Europe Conference Exhibition (DATE), 2012, pp. 983–987 (2012)
11. Beraldi, P., Ruszczynski, A.: Beam search heuristic to solve stochastic integer problems under probabilistic constraints. Eur. J. Oper. Res. **167**(1), 35–47 (2005)

12. Bertsimas, D., Nohadani, O.: Robust optimization with simulated annealing. J. Glob. Optim. **48**, 323–334 (2010). doi:10.1007/s10898-009-9496-x
13. Bertsimas, D., Sim, M.: The price of robustness. Oper. Res. **52**(1), 35–53 (2004). doi:10.1287/opre.1030.0065
14. Bilsen, G., Engels, M., Lauwereins, R., Peperstraete, J.: Cyclo-static data flow. In: 1995 International Conference on Acoustics, Speech, and Signal Processing (ICASSP-95), vol. 5, pp. 3255–3258 (1995)
15. Bilsen, G., Engels, M., Lauwereins, R., Peperstraete, J.: Cycle-static dataflow. IEEE Trans. Signal Process. **44**(2), 397–408 (1996)
16. Bodin, B., Munier Kordon, A., Dupont de Dinechin, B.: Periodic schedules for cyclo-static dataflow. In: ESTImedia, pp. 105–114 (2013)
17. Burns, A., Bernat, G., Broster, I.: A probabilistic framework for schedulability analysis. In: Proceedings of the Third International Conference on Embedded Software (EMSOFT 2003), pp. 1–15 (2003)
18. Calafiore, G., Campi, M.: Uncertain convex programs: randomized solutions and confidence levels. Math. Program. **102**, 25–46 (2005)
19. Calafiore, G., Campi, M.: The scenario approach to robust control design. IEEE Trans. Autom. Control **51**(5), 742–753 (2006)
20. Campi, M., Garatti, S.: A sampling-and-discarding approach to chance-constrained optimization: feasibility and optimality. J. Optim. Theory Appl. **148**(2), 257–280 (2011)
21. Carpov, S., Cudennec, L., Sirdey, R.: Throughput constrained parallelism reduction in cyclo-static dataflow applications. Procedia Comput. Sci. **18**, 30–39 (2013)
22. Carpov, S., Sirdey, R., Carlier, J., Nace, D.: Memory bandwidth-constrained parallelism dimensioning for embedded many-core microprocessors. In: CPAIOR10 Workshop on Combinatorial Optimization for Embedded System Design, Bologna, Italy (2010)
23. Charnes, A., Cooper, W.: Chance-constrained programming. Manag. Sci. **6**, 73–89 (1959)
24. Chen, X., Sim, M., Sun, P.: A robust optimization perspective on stochastic programming. Oper. Res. **55**(6), 1058–1071 (2007)
25. Dantzig, G.: Linear programming under uncertainty. Manag. Sci. **1**(3–4), 197–206 (1955)
26. Dentcheva, D., Prékopa, A., Ruszczynski, A.: Concavity and efficient points of discrete distributions in probabilistic programming. Math. Program. **89**, 55–77 (2000)
27. Dentcheva, D., Prékopa, A., Ruszczynski, A.: Bounds for probabilistic integer programming problems. Discret. Appl. Math. **124**(1–3), 55–65 (2002)
28. Devarakonda, M., Iyer, R.: Predictability of process resource usage: a measurement-based study on unix. Softw. Eng. IEEE Trans. **15**(12), 1579–1586 (1989)
29. Diaz, J., Garcia, D., Kanghee, K., Chang-Gun, L., Lo Bello, L., Lopez, J., Sang Lyul, M., Mirabella, O.: Stochastic analysis of periodic real-time systems. In: Real-Time Systems Symposium (RTSS 2002), pp. 289–300 (2002)
30. Dupont de Dinechin, B., Ayrignac, R., Beaucamps, P., Couvert, P., Ganne, B., Guironnet de Massas, P., Jacquet, F., Jones, S., Morey Chaisemartin, N., Riss, F., Strudel, T.: A clustered manycore processor architecture for embedded and accelerated applications. In: HPEC, pp. 1–6 (2013)
31. Efron, B., Tibshirani, R.: An Introduction to the Bootstrap. Taylor and Francis, Oxford (1994)
32. El Ghaoui, L., Oustry, F., Lebret, H.: Robust solutions to uncertain semidefinite programs. Siam J. Optim. **9**(1), 33–52 (1998)
33. Freund, R.F.: Optimal selection theory for superconcurrency. In: Proceedings of the 1989 ACM/IEEE Conference on Supercomputing, Supercomputing'89, pp. 699–703. ACM, New York (1989)
34. Gaivoronski, A., Lisser, A., Lopez, R., Xu, H.: Knapsack problem with probability constraints. J. Global Optim. **49**, 397–413 (2011)
35. Galea, F., Sirdey, R.: Méthode de cadencement d'applications flot de données cyclostatiques. Technical report, CEA LIST/DACLE/10-070 (2010)
36. Galea, F., Sirdey, R.: A parallel simulated annealing approach for the mapping of large process networks. In: IPDPS Workshop, pp. 1787–1792 (2012)

37. Gustafson, J.: Reevaluating Amdahl's law. Commun. ACM **31**(5), 532–533 (1988)
38. Hanen, C., Munier, A.: Cyclic scheduling on parallel processors: an overview. In: Chrétienne, P., Coffman, E.G., Lenstra, J.K., Liu, Z. (eds.) Scheduling Theory and Its Applications. Wiley, New York (1994)
39. Hennessy, J.L., Patterson, D.A.: Computer Architecture: A Quantitative Approach, 3rd edn. Morgan Kaufmann Publishers Inc., San Francisco (2003)
40. Henrion, R., Strugarek, C.: Convexity of chance constraints with independent random variables. Comput. Optim. Appl. **41**(2), 263–276 (2008)
41. Hong, L., Yang, Y., Zhang, L.: Sequential convex approximations to joint chance constrained programs: a Monte Carlo approach. Oper. Res. **59**(3), 617–630 (2011)
42. Howard, J., Dighe, S., Hoskote, Y., Vangal, S., Finan, D., Ruhl, G., Jenkins, D., Wilson, H., Borkar, N., Schrom, G., Pailet, F., Jain, S., Jacob, T., Yada, S., Marella, S., Salihundam, P., Erraguntla, V., Konow, M., Riepen, M., Droege, G., Lindemann, J., Gries, M., Apel, T., Henriss, K., Lund-Larsen, T., Steibl, S., Borkar, S., De, V., Van Der Wijngaart, R., Mattson, T.: A 48-core ia-32 message-passing processor with dvfs in 45 nm cmos. In: IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC), pp. 108–109 (2010)
43. Iverson, M., Ozguner, F., Follen, G.: Run-time statistical estimation of task execution times for heterogeneous distributed computing. In: Proceedings of 5th IEEE International Symposium on High Performance Distributed Computing, pp. 263–270 (1996)
44. Kahn, G.: The semantics of simple language for parallel programming. In: IFIP Congress, pp. 471–475 (1974)
45. Khokhar, A.A., Prasanna, V.K., Shaaban, M.E., Wang, C.L.: Heterogeneous computing: challenges and opportunities. Computer **26**(6), 18–27 (1993)
46. Klopfenstein, O.: Optimisation robuste de réseaux de télécommunications. Ph.D. thesis, Orange Labs, Laboratoire Heudiasyc, UMR CNRS 6599, Université de Technologie de Compiègne (2008)
47. Lee, E., Parks, T.: Dataflow process networks. Proc. IEEE **83**(5), 773–801 (1995)
48. Lee, J., Shin, I., Easwaran, A.: Online robust optimization framework for QoS guarantees in distributed soft real-time systems. In: Proceedings of the tenth ACM International Conference on Embedded Software, EMSOFT'10, pp. 89–98. New York (2010)
49. Lemerre, M., David, V., Aussagues, C., Vidal-Naquet, G.: Equivalence between schedule representations: theory and applications. In: Real-Time and Embedded Technology and Applications Symposium, RTAS'08, pp. 237–247. IEEE (2008). doi:10.1109/RTAS.2008.17
50. Li, P., Wendt, M., Wozny, G.: Robust model predictive control under chance constraints. Comput. Chem. Eng. **24**(2–7), 829–834 (2000)
51. Lombardi, M., Milano, M., Ruggiero, M., Benini, L.: Stochastic allocation and scheduling for conditional task graphs in multiprocessor systems-on-chip. J. Sched. **13**, 315–345 (2010)
52. Loughlin, D.H., Ranjithan, S.: Chance-constrained genetic algorithms. In: GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 369–376 (1999)
53. Louise, S.: The future of programming embedded systems: methods for timing or for performance support. HDR, EDIPS, Paris-Sud, France (2014)
54. Luedtke, J., Ahmed, S.: A sample approximation approach for optimization with probabilistic constraints. SIAM J. Optim. **19**(2), 674–699 (2008)
55. Manolache, S., Eles, P., Peng, Z.: Memory and time-efficient schedulability analysis of task sets with stochastic execution time. In: 24th Euromicro Conference on Real-Time Systems, pp. 0019 (2001)
56. Mazouz, A., Touati, S.A.A., Barthou, D.: Study of variations of native program execution times on multi-core architectures. In: CISIS, pp. 919–924 (2010)
57. Miller, B., Wagner, H.: Chance constrained programming with joint constraints. Oper. Res. **13**(6), 930–945 (1965)
58. Nemirovski, A., Shapiro, A.: Convex approximations of chance constrained programs. SIAM J. Optim. **17**(4), 969–996 (2006)
59. Optimization, G.: Gurobi—state of art mathematical programming solver (2014). http://www.gurobi.com/products/gurobi-optimizer/gurobi-overview

60. Pagnoncelli, B.K., Ahmed, S., Shapiro, A., Pardalos, P.M.: Sample average approximation method for chance constrained programming: theory and applications. J. Optim. Theory Appl. **142**, 399–416 (2009)
61. Pal, B., Gupta, S., Chakraborti, D.: A genetic algorithm based stochastic simulation approach to chance constrained interval valued multiobjective decision making problems. In: 2010 International Conference on Computing Communication and Networking Technologies (ICCCNT), pp. 1–7 (2010)
62. Prékopa, A.: Stochastic Programming. Kluwer Academic, Dordrecht (1995)
63. Reistad, B., Gifford, D.K.: Static dependent costs for estimating execution time. SIGPLAN Lisp Pointers VI **VII**(3), 65–78 (1994)
64. Rockafellar, R.T., Uryasev, S.: Optimization of conditional value-at-risk. J. Risk **2**, 21–41 (2000)
65. Soyster, A.L.: Convex programming with set-inclusive constraints and applications to inexact linear programming. Oper. Res. **21**(5), 1154–1157 (1973)
66. Stan, O., Sirdey, R., Carlier, J., Nace, D.: A heuristic algorithm for stochastic partitioning of process networks. In: Proceedings of the 16th IEEE International Conference on System Theory, Control and Computing (ICSTCC), pp. 1–6 (2012)
67. Stan, O., Sirdey, R., Carlier, J., Nace, D.: A GRASP for placement and routing of dataflow process networks on manycore architectures. In: 3PGCIC, pp. 219–226 (2013)
68. Stan, O., Sirdey, R., Carlier, J., Nace, D.: The robust binomial approach to chance-constrained optimization problems with application to stochastic partitioning of large process networks. J. Heuristics **20**(3), 261–290 (2014)
69. Tanner, M.W., Beier, E.B.: A general heuristic method for joint chance-constrained stochastic programs with discretely distributed parameters (2007). http://www.optimization-online.org/DB_FILE/2007/08/1755.pdf
70. UTK, ORNL: Netlib repository (2014). http://www.netlib.org/
71. Wilhelm, R., Engblom, J., Ermedahl, A., Holsti, N., Thesing, S., Whalley, D., Bernat, G., Ferdinand, C., Heckmann, R., Mitra, T., Mueller, F., Puaut, I., Puschner, P., Staschulat, J., Stenström, P.: The worst-case execution-time problem: overview of methods and survey of tools. ACM Trans. Embed. Comput. Syst. **7**(3), 36:1–36:53 (2008)
72. Xu, H., Caramanis, C., Mannor, S.: Optimization under probabilistic envelope constraints. Oper. Res. **60**(3), 682–699 (2012)
73. Yang, J., Ahmad, I., Ghafoor, A.: Estimation of execution times on heterogeneous supercomputer architectures. In: International Conference on Parallel Processing, (ICPP 1993), vol. 1, pp. 219–226 (1993)