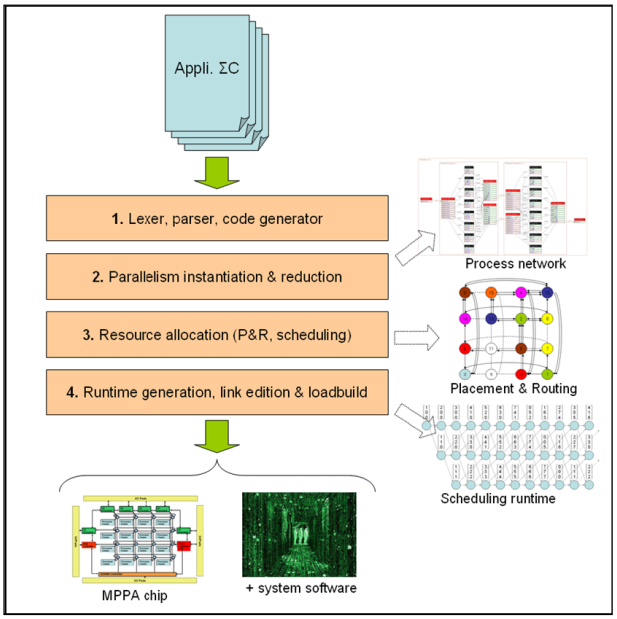
Par **Oana STAN**

Placement of tasks under uncertainty on massively multicore architectures

Thèse présentée
pour l'obtention du grade
de Docteur de l'UTC



Soutenue le 15 novembre 2013
Spécialité : Technologies de l'Information et des Systèmes

D2116

UNIVERSITÉ DE TECHNOLOGIE DE COMPIÈGNE

PLACEMENT OF TASKS UNDER UNCERTAINTY ON MASSIVELY MULTICORE ARCHITECTURES

THÈSE DE DOCTORAT

pour l'obtention du grade
de docteur de l'Université de Technologie de Compiègne
spécialité Technologies de l'Information et des Systèmes

présentée et soutenue publiquement
par
Mme. Oana STAN

le 15 novembre 2013

devant le jury composé de :

Directeurs de thèse :	Jacques Carlier	Professeur, Université de Technologie de Compiègne
	Dritan Nace	Professeur, Université de Technologie de Compiègne
Rapporteurs :	Alix Munier	Professeur, Université Paris VI
	Walid Ben-Ameur	Professeur, Institut TELECOM Sud-Paris
Encadrant :	Renaud Sirdey	Ingénieur-chercheur, CEA, LIST

To my father

Acknowledgements

First of all, I want to address my grateful acknowledgments to Mr. Jacques CARLIER and Mr. Dritan NACE, professors at Université de Technologie de Compiègne, for directing my research work and guiding my steps with their eminent expertise in the field of combinatorial optimization. Thanks to their constant and careful supervision, I learned the good practices required in order to fulfill a high quality research and I was able to finish my Phd dissertation in time without any further delay.

My foremost appreciation also goes to Mr. Renaud SIRDEY, my Phd supervisor at CEA, for his permanent support, his endless ideas helping me progress during these last three years, for the confidence he had in me and for his contagious enthusiasm for Science.

I wish to thank Mrs. Alix MUNIER, professor at Université Paris 6, and Mr. Walid BEN-AMEUR, professor at Telecom SudParis, for honoring me in accepting to report this thesis and for their valuable remarks and suggestions. Also, I want to address my thanks to Mr. Aziz MOUKRIM, professor at Université de Technologie de Compiègne, for accepting to be president of the examining committee.

I want to address my gratitude to all those who made possible this work.

I especially think of the members of LaSTRE laboratory, from CEA: the coffee breaks, the discussions, the advices, the after-works together not only made this an enjoyable professional experience, they were also important for my personal growth and enrichment.

A grateful thought goes to my Phd old fellows Nicolas, Ilias and Thomas for the good moments during the lunches we had together at Resto 1...

Also, many thanks to my deskmates from "the Phd cage" at NanoInnov - Vincent, Simon, Safae, Karl, Moez, the two Amira - for all their help, for all the sweets we shared and for bearing with me and with my continuous complains - "J'en ai marre", "Ca marche pas!!!", "Oh-la-la!" just to name a few...

A special thought for "The Three Musketeers"- Pascal, Sergiu and Julien - thanks to their jokes and their support, overcoming the inherent moments difficult of a Phd was less complicated.

I want to express my acknowledgments to all those who helped improve this thesis, by re-readings or by providing experimental data: Safae, Vincent, Karl, Pascal, Julien, Sergiu (again) and also Cyril, Loïc, Paul and Lilia.

Many thanks to my friends outside work - Linda, Corina, Mihaela & Marius, Valentin, Anca - due to their presence, I managed to keep my head outside the box of my Phd topic, see the outside world and realize that there is a life out there.

Finally, my heartfelt thanks my family, for their comforting words and their support, from my earliest childhood until now. It's also thanks to them that I became what I am now and I succeeded to get so far. I wish to thank my mother for always being beside me, my aunt (Monica) and my grandmothers for their pieces of advice as well as my little cousins (Vladut and Alex) for their unconditional love. I would like to pay tribute to my father, which was the first one to persuade me to follow this path and which would have been happy to live this moment.

Last, but not least, my warmest thanks to Andrei, my constant supporter, for his kindness and his love, for his help and for the lost weekends, and for simply standing beside me, while facing the ups and downs of being a Phd student.

Résumé

Ce travail de thèse de doctorat est dédié à l'étude de problèmes d'optimisation combinatoire du domaine des architectures massivement parallèles avec la prise en compte des données incertaines tels que les temps d'exécution. On s'intéresse aux programmes sous contraintes probabilistes dont l'objectif est de trouver la meilleure solution qui soit réalisable avec un niveau de probabilité minimal garanti.

Une analyse qualitative des données incertaines à traiter (variables aléatoires dépendantes, multimodales, multidimensionnelles, difficiles à caractériser avec des lois de distribution usuelles), nous a conduit à concevoir une méthode qui est non paramétrique, intitulée « approche binomiale robuste ». Elle est valable quelle que soit la loi jointe et s'appuie sur l'optimisation robuste et sur des tests d'hypothèse statistique. On propose ensuite une méthodologie pour adapter des algorithmes de résolution de type approchée pour résoudre des problèmes stochastiques en intégrant l'approche binomiale robuste afin de vérifier la réalisabilité d'une solution. La pertinence pratique de notre démarche est enfin validée à travers deux problèmes issus de la compilation des applications de type flot de données pour les architectures manycore.

Le premier problème traite du partitionnement stochastique de réseaux de processus sur un ensemble fixé de nœuds, en prenant en compte la charge de chaque nœud et les incertitudes affectant les poids des processus. Afin de trouver des solutions robustes, un algorithme par construction progressive à démarrages multiples a été proposé ce qui a permis d'évaluer le coût des solutions et le gain en robustesse par rapport aux solutions déterministes du même problème.

Le deuxième problème consiste à traiter de manière globale le placement et le routage des applications de type flot de données sur une architecture clustérisée. L'objectif est de placer les processus sur les clusters en s'assurant de la réalisabilité du routage des communications entre les tâches. Une heuristique de type GRASP a été conçue pour le cas déterministe, puis adaptée au cas stochastique clusterisé.

Abstract

This PhD thesis is devoted to the study of combinatorial optimization problems related to massively parallel embedded architectures when taking into account uncertain data (e.g. execution time). Our focus is on chance constrained programs with the objective of finding the best solution which is feasible with a preset probability guarantee.

A qualitative analysis of the uncertain data we have to treat (dependent random variables, multimodal, multidimensional, difficult to characterize through classical distributions) has lead us to design a non parametric method, the so-called "robust binomial approach", valid whatever the joint distribution and which is based on robust optimization and statistical hypothesis testing. We also propose a methodology for adapting approximate algorithms for solving stochastic problems by integrating the robust binomial approach when verifying for solution feasibility. The practical relevance of our approach is validated through two problems arising in the compilation of dataflow application for manycore platforms.

The first problem treats the stochastic partitioning of networks of processes on a fixed set of nodes, by taking into account the load of each node and the uncertainty affecting the weight of the processes. For finding stochastic solutions, a semi-greedy iterative algorithm has been proposed which allowed measuring the robustness and cost of the solutions with regard to those for the deterministic version of the problem.

The second problem consists in studying the global placement and routing of dataflow applications on a clusterized architecture. The purpose being to place the processes on clusters such that it exists a feasible routing, a GRASP heuristic has been conceived first for the deterministic case and afterwards extended for the chance constrained variant of the problem.

Contents

Introduction

> *As far as the laws of mathematics refer to reality, they are not certain; and as far as they are certain, they do not refer to reality.*
>
> — Albert Einstein

The world we are living in is subject to permanent changes, where uncertainty is encountered each day, everywhere, under various flavors, affecting individuals as well as collective consciousness. We can choose to ignore it, which is the most convenient path, or we can embrace the uncertainty surrounding us, accept it and take it into account in our quest of the "*Truth*".

Unfortunately, when making decisions, uncertainty is often neglected and the best decision is selected by applying classical methods from the combinatorial optimization field, without worrying about the variations of the data. Combinatorial optimization, one of the most popular branches of mathematics over the last half of century, with a widespread area of application from logistics and transportation to planning and scheduling, finance or engineering design, allows to solve combinatorial problems by finding the best solution from a set of finite but potentially enormous size.

Dantzig, one of the greatest pioneers in the fields of operations research and management science, considered optimization under uncertainty amongst the most promising and important areas of combinatorial optimization. A fervent promoter of the subject of stochastic optimization throughout the last thirty years, he emphasized during an interview in 2000 [1]:

> *Planning under uncertainty. This, I feel, is the real field that we should be all working in. Those of us who were doing the planning right from the very beginning understood that the real problem was to be able to do planning under uncertainty. The mathematical models that we put together act like we have a precise knowledge about the various things that are happening and we don't.*

Embedded systems design is one of the major domains of computer industry for which applying optimization under uncertainty seems legitimate. With the arrival of the first manycore embedded architectures, compilation of applications targeting these platforms has to be efficient in order to fully and reasonably exploit the capacity constraints existing on a reduced number of interdependent resources (CPU,

1. Irvin Lustig. In his own voice - Interview with G. Dantzig "Planning under uncertainty": https://www2.informs.org/History/dantzig/in_interview_irv10.htm.

bandwidth, memory, etc.). As such, the resolution of related optimization problems via operations research techniques has spread from compilers backend throughout the overall compilation process. To cite only a few, we can mention the instruction scheduling and buffer problems or more recent optimization problems such as the dimensioning of communication buffers or the partitioning and placement of process networks. Or, even if a common characteristic of these problems is the presence of uncertain data such as execution times or network latencies, the research focusing on applying techniques from optimization under uncertainty is still at the beginning.

Our works are situated in this context, of optimizing under uncertainty for the compilation of applications for embedded manycore.

Hence, this dissertation deals with the application of the most appropriate methods from the field of optimization under uncertainty to the resolution of combinatorial problems arising in compilation for massively multi-core embedded systems. Such platforms, composed of several parallel cores, a number of memory mechanisms and an infrastructure connecting all the components together, require applications that can efficiently take advantage of the available resources and of the parallelism. One viable way of programming applications for the new generation of embedded architectures is based on dataflow models, in which applications are expressed as set of tasks communicating through FIFO channels.

Our applicative work focuses mainly on the partitioning, placement and routing of dataflow applications under the assumption of uncertain data (in particular unitary execution times). These combinatorial optimization problems related to resource allocation are part of the several steps composing the compilation process of a dataflow application.

Since the general topic of the present study has been introduced, let us now mention some of the research questions which drew our attention and guided our research path:

- For a given optimization problem from embedded domain, what is the benefit of taking into account uncertainty instead of solving the deterministic version?
- How can be captured, analyzed and expressed the uncertainty occurring in compilation of applications for embedded systems?
- What are the most relevant models and resolution techniques of uncertain optimization when dealing with combinatorial problems from the embedded domain?
- Once found, how these resolution methods for optimizing under uncertainty can be applied in an operational manner for solving the application case studies?

Throughout our present dissertation, we will show the importance of optimization under uncertainty. Moreover, applying it for solving problems for *soft real-time embedded systems* can be beneficial and even mandatory in some cases in order to find robust solutions. For these types of applications and a given optimization problem, we are looking for the best solution which is guaranteed to be feasible with a target probability (for example, the best solution which is feasible with a probability of 0.9). When designing soft real-time systems for a large class of applications (audio encoding, multimedia, telecommunications, etc.), breaking requirements, even if not desirable, is acceptable if it happens with a sufficiently low probability since the

overall required Quality of Service (QoS) is not compromised. A typical example consists of a possible loss of some frames from a group of pictures in an MPEG encoding. As such, by accounting for data variations when dimensioning our system we can avoid oversizing, expensive in terms of hardware, or undersizing, expensive in terms of reliability. Also, we could estimate the robustness of an already dimensioned system and find scenarios which are not feasible or not acceptable. We are not considering the case of safety-critical systems (nuclear plant control and command, automotive, avionics), for which the design has to be based on the worst-case scenario since any break of the requirements cannot be accepted due to the highly potential risks.

For answering the other questions, which are in some sense inter-related, we began by analyzing the sources of uncertainty affecting data from optimization problems in an embedded environment and in particular the execution times. As the state of art and some basic qualitative examples have pointed out, execution times are difficult to characterize and do not follow classical distributions laws. At best, we can affirm that they are random variables of a bounded support, dependent and multidimensional.

The above conclusions have limited our choice to resolution approaches which can be applied for embedded field without making simplifying or making erroneous assumptions about the uncertain data. Other prerequisite we are focusing on is the ability to find solutions which are guaranteed with a minimal probability threshold. Also, considering the complexity of the combinatorial problems and the size of the instances in the compilation process, we have to be able to fit the uncertainty treatment into approximation methods, heuristics or metaheuristics.

Between the existing optimization approaches we are aware of, the only one making almost no assumptions on the uncertain data is the one based on scenarios. Its original form is too conservative, since it searches for solutions satisfying all the scenarios. Therefore, we preferred to extend it and conceive a new method, the robust binomial approach, which can find solutions which are guaranteed to hold for a minimal required probability with a high confidence level. The only assumption we are making is the existence of a sample with a sufficient size of independent and identically distributed observations for the uncertain data (which can exhibit dependencies).

Moreover, the robust binomial approach can be easily integrated within an heuristic or metaheuristic in the oracle deciding the feasibility of a potential solution. We, thus, proposed an overall framework for solving problems with probabilistic constraints using heuristics and the robust binomial approach. Also, assuming an approximate resolution algorithm had already been implemented to solve a problem in the deterministic case, we thought of the necessary steps for adapting (meta)heuristics to the stochastic version of the same problem.

We then applied the robust binomial approach to the stochastic partitioning of process networks for which we consider uncertain weights for the processes. For the other application case, the joint placement and routing of dataflow applications, we first design a GRASP (Greedy Randomized Adaptative Search Procedure) treating the deterministic case since this problem has not been yet treated in a global manner. Afterwards, we attacked the stochastic problem supposing the tasks have uncertain

weights and integrating the robust binomial approach into the GRASP.

The present manuscript, resuming our contribution, is organized into this introduction, four chapters and a conclusion. (Part of the work of this thesis has also been subject of several publications, referenced on a separated page.) Let us now give a brief overview of each chapter.

Chapter 1 introduces the context and gives more details about the motivations of our research. The first two sections present the emergence of massively parallel embedded systems, the main hardware components of a manycore architecture as well as the difficulties in developing applications for this kind of systems. Then, we provide an overview of the dataflow paradigm, an alternative to sequential programming which seems more appropriate for programming manycore applications. Sigma-C, a dataflow programming model and language conceived by the CEA and the associated compilation process, are the subject of the following section. We conclude the chapter with a motivation section, in which we also provide a short qualitative analysis of execution times, one of the main uncertainty source for embedded systems, related to optimization problems.

Chapter 2 is dedicated to optimization under uncertainty and, in particular, to the robust binomial approach, the non-parametric resolution method we conceived to cope with uncertain data difficult to model or to characterize. Some general considerations about the different techniques for optimization under uncertainty, such as stochastic programming and robust optimization, are given in the first part. The second section presents the general structure of the type of problems we are interested in, joint chance constrained programs for which we search the best solution feasible for all constraints with a minimal probabilistic guarantee. We show the difficulties in solving chance constrained programs and we discuss existing work, classified in convexity studies, robust optimization methods, sampling techniques and (meta)heuristics. The next section explains more in details the robust binomial approach, the statistics beyond it, the way it can be applied for solving a chance constrained problem and the methodology for integrating it into an existing metaheuristic. The last part proposes some possible extensions of the robust binomial approach to more general problems (with more than one initial probability level requirement or uncertain objective function).

In Chapter 3 we study the stochastic problem of partitioning networks of processes onto a fixed number of nodes. Given a dataflow application, the objective is to assign the tasks to a fixed number of processors in order to minimize the total communications (which correspond to minimizing communications between processors) while respecting the capacity of each processor in terms of resources (memory footprint, core occupancy etc.). An extension of the Node Capacitated Graph Partitioning problem, this application case is known to be NP-hard. The stochastic version we study here is the partitioning for which the weights of the tasks are uncertain resources (processor occupancy, memory footprint) and thus, we have to solve the chance constrained program for which the capacity of each processor is respected for each resource with a minimal probability target. For clarification purposes, a preliminary section introduces the concepts and the greedy heuristic on which our resolution algorithm is based. The adaptations made for integrating the robust binomial approach into the existing semi-greedy heuristic

are presented in a separated section and experimental results are given at the end. The tests performed have two main purposes: showing the importance of taking into account data variations and measuring "the price of robustness" compared to the deterministic version. Running under the same assumptions as the stochastic algorithm, the method for the deterministic problem is unable to find feasible solutions on a large number of cases. As for the "price of robustness", the stochastic solutions are consistent with those found in the deterministic case. More important, they are guaranteed to hold with a high probabilistic and confidence levels.

The other application case, the joint placement and routing problem, is studied in Chapter 4 from both the deterministic and stochastic perspectives. The purpose is 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 (NoC), respecting the maximal available bandwidth. Each of the two subproblems, tasks placement and respectively routing, are NP-hard and treating them together, in a single step, could be convenient mainly in the case of applications with high bandwidth demands (such as multimedia or computer vision), for which the bandwidth resources of the NoC can become critical. Moreover, treating them separately, in a sequential manner, can lead to placements for which the routing is impossible subsequently. The introductory sections present the formal description of the problem and existing deterministic mapping approaches, static or dynamic. As for the stochastic case, we give some related works but to the best of our knowledge, the exact same problem has not been yet addressed in the literature. The next section proposes a GRASP (Greedy Randomized Adaptative Search Procedure) heuristic for solving the deterministic version. In order to solve the chance-constrained problem when the resources of the tasks are uncertain, we extended the GRASP via the robust binomial approach and the necessary changes are given in another section. Extensive computational results on synthetic benchmarks and a real application from the image processing field are given at the end for the deterministic case as well as for the stochastic one.

List of Publications

- O. Stan, R. Sirdey, J. Carlier, D. Nace. "L'apport de l'optimisation sous incertitudes pour les systèmes temps-réel embarqués." (Ecole de temps réel - ETR11), Brest, France. 2011

- O. Stan, R. Sirdey, J. Carlier, D. Nace. "A Heuristic Algorithm for Stochastic Partitioning of Process Networks." (Proceedings of the 16th IEEE International Conference on System Theory, Control and Computing - ICSTCC), Sinaia, Romania. 2012

- O. Stan, R. Sirdey, J. Carlier, D. Nace. "A GRASP for placement and routing of dataflow process networks on manycore architectures." (Proceedings of the 8th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing - 3PGCIC), Compiègne, France. 2013

- O. Stan, R. Sirdey, J. Carlier, D. Nace. "The Robust Binomial Approach to chance-constrained optimization problems with application to stochastic partitioning of large process networks." (Submitted to "Journal of Heuristics"), 2012

Research Context and Motivations

Contents

1.1 Introduction

One of the key requirements when designing embedded systems solutions nowadays is the performance in terms of computer power for all supported applications. The evolution of latest applications like video and image processing based on sophisticated compression and decompression algorithms (MPEG 4, H.264, etc.), 3D video games, scientific computing or data security determines a demand in computer power ten to a hundred times superior to that of a few years ago and even make the performance requirements for these embedded systems exceed the abilities of most desktop computers.

Unfortunately, over the last past years, it has become obvious that the performance offered by traditional sequential single core processors has not kept step with the demand. Even if, according to the original Moore's law [121], the number of transistors which can be placed on a single chip has continued to double every two years, the performance of practical computing does not follow the same exponential growth rate [1]. Several causes exist for this phenomenon, called Moore's gap: poor returns from single CPU mechanisms such as caching and pipelining, the power envelopes (both active and leakage related), wire delay, etc.

Between the viable solutions to improve performance we can cite the conversion of additional transistors into computing power at relatively low clock frequency by designing and using efficient parallel processing systems.

1. The Pentium 4, first implemented with the same technology as the Pentium 3 (0.18-micron) is a popular example demonstrating the break in performance scaling. Despite that Pentium 4 had 50% more transistors than Pentium 3, its performance, based on the SPECint 2000, was only 15% greater. (http://www.embedded.com/design/mcus-processors-and-socs/4007064/Going-multicore-presents-challenges-and-opportunities)

The present multi-core architectures are achieving more performance by the use of several processing elements (roughly a dozen) and the next generation of manycore chips is even more powerful, containing hundreds if not thousands of cores. As such, we are entering into a manycore era in which the updated Moore's law states that the number of cores on a chip doubles every two years. Fig. 1.1 [108] captures the general exponential evolution trend of the number of individual computing units according to the release years of chips.

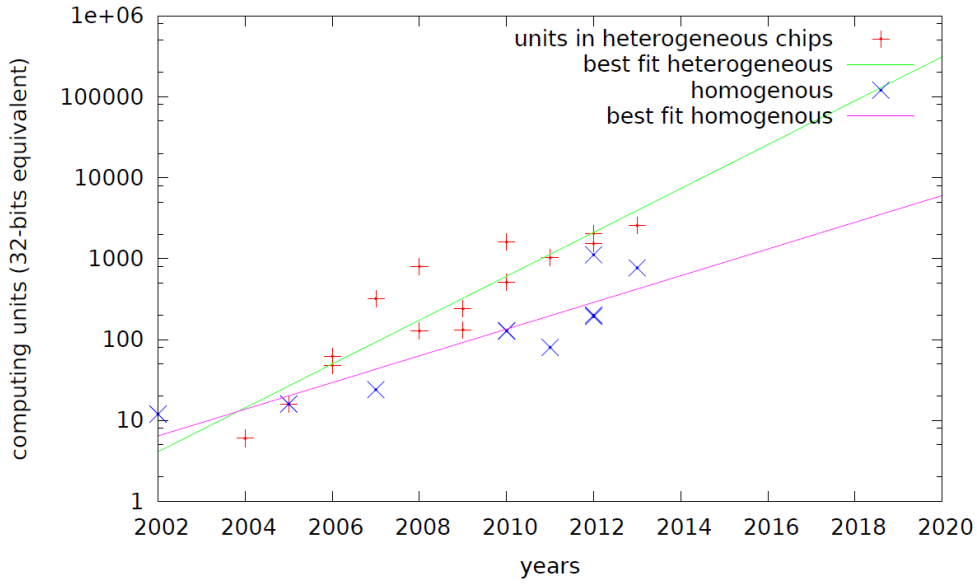


Figure 1.1: Number of individual processing units in heterogeneous chips (e.g. AMD, NVidia graphics, IBM Cell BE, etc) and homogeneous chips (e.g. Intel Xeon, IMB Power, STM, Tilera, Kalray, etc) [108]

However, in order to efficiently exploit the parallelism [2] and to fully take advantage of the computing power these manycore may provide, their design requires new programming and execution paradigms as well as innovative compilation technologies. After a brief description of the hardware architecture of a type of embedded manycores, the next subsections are describing the challenges manycore have to face and some appropriate programming and execution models allowing to efficiently exploit the parallelism.

1.2 Massively parallel embedded systems

1.2.1 Flynn's classification

Flynn's macroscopic classification of computing systems [67], realized in function of the possible interaction patterns between instructions and data, distinguishes between four different groups of machines: Single Instruction Single Data (SISD), Single Instruction Multiple Data (SIMD), Multiple Instruction Single Data (MISD)

2. Most existing applications developed in the last decades were conceived for a sequential execution.

and Multiple Instruction Multiple Data (MIMD). Between these categories, the first one corresponds to a casual sequential processor, only the last three making parallel execution possible. As such, 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 non-structured data and conditional or data-dependent algorithms, the MIMD is a more usual implementation of the many-core concept.

Furthermore, according to the memory organization, existing MIMD can be grouped into DMM (Distributed Memory Machines) and SMM (Shared Memory Machines). In the distributed memory machines, each processing element has its own local memory to which it has direct access. The access to other processor's local memory is a message-passing operation performed using the interconnection network. In a shared memory system, the processors have a common memory space and communications between them are realized by reading and writing data to a shared address in the memory. One of the advantages of the SMM over a DMM is the ease of communication via the shared memory and without data replication. However, due to bandwidth limits, the number of processors in a SMM is limited because using a larger number of processors results in an increase in the access times at effective memory bus. As such, the SMMs can provide more computing power while DMMs are more scalable.

1.2.2 Manycore architectures

A massively multi-core (manycore) processor is a parallel computing system, composed of a number of processing cores, a mix of local and shared memory, distributed global memory or multilevel cache hierarchy and an infrastructure for inter-cores communication. The efficiency in such a system is determined by a high scalability and computing power.

In a clustered massively multi-core chip, the processing elements are organized in clusters, interconnected via a Network-On-Chip (NoC). This type of architecture represents a DMM in which the nodes are not single processors, but SMMs. As such, it provides a solution for the problem of scalability of the SMMs, for which it is difficult to exceed 32 processors [143], and for the performance issues of the DMMs. As illustrated in Fig.1.2 [33], each cluster contains more processing elements, with a processing core and a private cache memory, as well as a shared memory space. The entire processor is connected through a memory access controller to an additional external memory. The external memory is used for storing application data and instructions. The processors belonging to the same cluster exchange their data efficiently, using the local shared memory space. For communication of processors on different clusters, data transfers are assured by the Network-On-Chip. In order to transfer data between clusters in an effective manner, the NoC must provide enough bandwidth. Therefore, an important factor in designing efficient manycore systems is the choice of an appropriate interconnection network.

An alternative to the basic communication structures in a System-On-Chip (e.g. traditional bus-based communication and dedicated link-to-link points), a well designed chip network can avoid performance bottleneck which appear in manycore

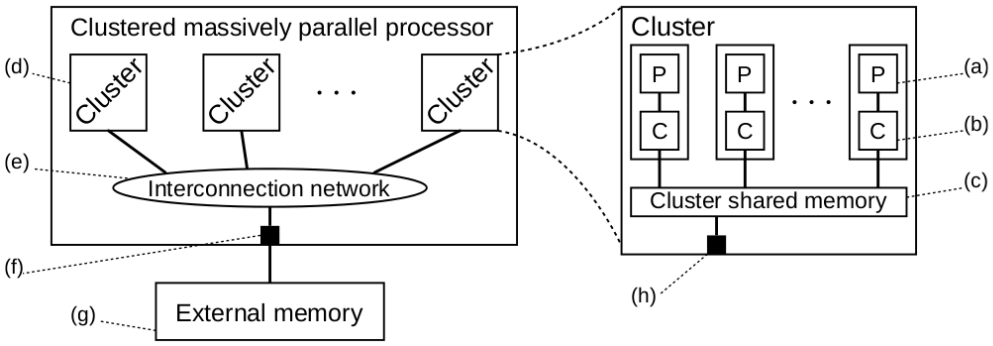


Figure 1.2: Clustered massively multi-core architecture(a - processor, b - processor cache, c - cluster shared memory, d - cluster, e - cluster interconnection network (NoC), f - external memory access controller, g - external memory, h - network adapter)[33]

applications. Basically, the main components of a NoC are the network adapters, the routing nodes and the links. The network adapters implement the interface by which the cores connect to the network, the routing nodes implement the routing strategy (route the data according to chosen protocols) and the links (logical or physical channels) connect the nodes and provide the raw bandwidth. The key factors defining an on-chip network are its topology and the routing protocol. The topology concerns the geometrical layout and describes the connectivity between routers and links while the protocol concerns the way data are moved across the NoC. Fig.1.3 shows a sample of 4-by-4 grid-based NoC and its fundamental components. For more details about the current research and practices concerning the NoC, we refer the reader to [21].

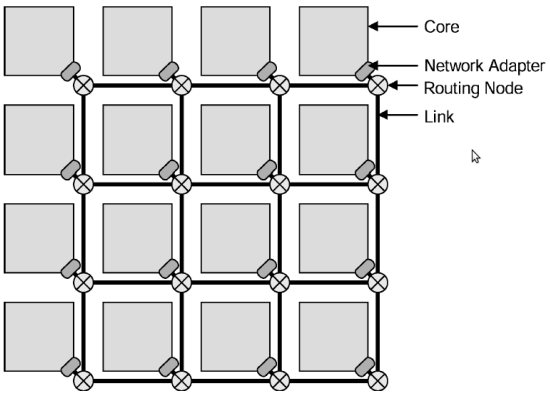


Figure 1.3: A structured 4×4 grid NoC and its components [21]

1.2.3 Challenges of programming for manycores

According to Gustafson's law [75], as more computing power becomes available, new classes of complex applications (e.g. software and cognitive radio, autonomous vehicles, virtual and augmented reality) emerge. However, programming these applications for manycore systems is a difficult task, since there are at least three

difficulties to overcome: handle limited and dependent resources (memory, NoC), be able to run correctly large parallel programs and efficiently exploit the underlying parallel architectures.

The first issue is already solved in existing embedded manycore like Kalray MPPA platforms or ST-Micro Storm using some kind of hierarchical memory architecture, with distributed memories close to the processing elements and a shared on-chip memory for communication with other clusters and the outside world.

For taking advantage of manycores architectures in terms of computing power, power consumption or development cost, one must be able to efficiently parallelize an application and thus, it demands a "good" decomposition of the program into tasks. Traditional imperative programming languages (C or Java) are based on a sequential von Neumann architecture and therefore they are inappropriate for writing effective parallel programs. As such, there is an increasing need of other programming paradigms for writing embedded manycore applications, that should satisfy the following properties [108]:

- Easy synchronization between program tasks. Since one of the most difficult and error-prone jobs in parallel programming is explicit synchronization, it is recommended to avoid synchronization structures or mask their use.
- Execution determinism. For constant entry dataset, computation results should be independent of the execution hazards coming, for example, from task scheduling and allocation.
- Possibility to easily integrate existing code. This is necessary in order to migrate or integrate legacy software developed in familiar languages for embedded programmers.

In [84], Jantsch identifies and divides un-timed models adapted to non critical embedded systems into two categories: models based on rendez-vous and dataflow process networks. In a rendez-vous model, tasks are modeled through concurrent sequential processes which communicate with each other only at certain synchronization points. Common examples of rendez-vous models are the CSP (Communicating sequential processes), the CCS (Calculus of Communicating Systems) or the model of communication used by the ADA language. In a dataflow process network, the tasks are modeled by sequential processes which exchange data through communications channels. As such, a dataflow program can be represented as a directed graph with the nodes representing the processes and the arcs representing the channels. In such models, inter-tasks synchronization is realized implicitly, via the data.

In the following sections, we are concentrating on dataflow models and languages and, in particular, Sigma-C.

1.3 Dataflow models and stream programming

Dataflow paradigm seems to be a good candidate for programming manycore applications, which satisfy most of the properties stated before. With the first models emerging in the early 1970s, dataflow languages provide an efficient and simple solution to express programs, which can be executed on a parallel architecture, without worrying about data synchronization. Since a history of the evolution

of dataflow computation models is beyond the scope of this chapter, we refer the interested reader to the following texts [16], [82].

There exist several formalisms for dataflow models different in their expressive power and the guarantees they provide. Before describing more in detail some of the most representative dataflow models, let us introduce some general features of dataflow programming.

1.3.1 Taxonomy

In a dataflow model, a program is described as a directed graph, consisting of nodes representing *tasks* (also named *actors*) and arcs that represent unidirectional communication channels. The actors of a dataflow application can be atomic or a hierarchically specified subgraph. Data, quantized into *tokens*, is carried exclusively through the channels, considered to be First-In-First-Out (FIFO) queues. A *token* consists of the smallest indivisible quantum of data traversing the channels. Since the communications channels are unbounded, they can potentially carry an infinite number of tokens. The flow passing through a channel is denoted as a *stream*.

The execution of a dataflow process is a sequence of *firings* or *evaluations*. When an actor is executed, it *consumes* a certain quantity of data tokens on its input channels and it *produces* a number of result tokens written on its output channels. Any node can fire (perform its computation) when all the required data is available on its input channels. Since there can be more actors firing in the same time, one of the interesting properties supported by dataflow languages is the *concurrency*. *Synchronization* between actors is realized exclusively via the data traversing the channels. Because the program execution depends on the availability of the data, these models belong to the family of *data-driven* models of computation.

The number of tokens produced or consumed may vary for each firing and is defined in the *firing rules* of an actor. An actor can have one or several firing rules and can be *static* or *dynamic*. For a dynamic actor, the choice of the firing rule is data-dependent and consequently, its behavior for future firings cannot be predictable. A static actor can have one or more firing rules while a dynamic actor has at least two.

In function of how the consumption and production of tokens and the firing rules are specified, dataflow computing models can be divided in plenty different classes. For a deeper insight description of dataflow computation models, we refer the reader to [73], [159], [84], [125]. The most representative dataflow models (cf. Fig. 1.4) are: Synchronous DataFlow (SDF), Cyclo-Static DataFlow (CSDF) and Dynamic DataFlow (DDF), with its subclass of Boolean DataFlow (BDF), all being particular classes of Kahn Process Networks (KPN).

1.3.2 Dataflow models

1.3.2.1 Dataflow process networks

A *dataflow process* [99] is a particular type of Kahn process consisting of a sequence of firings and a network of such processes is called a *dataflow process*

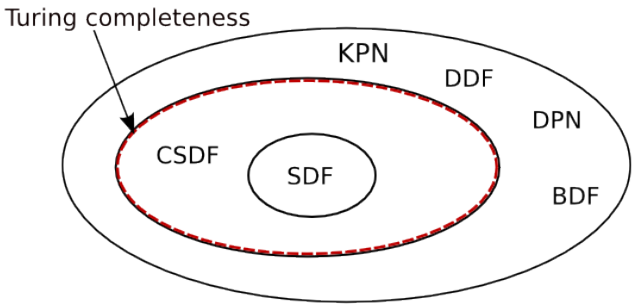


Figure 1.4: Representative dataflow models.

network (DPN). In a Kahn process network [87], which is the least constraint model, the network processes communicate with each other only via unbounded FIFO channels, with reading data from these channels being *blocking* and writing data asynchronous. As such, because the channel size is infinite, writing always succeeds and does not stall the processes while reading can be realized only from a non-empty channel and when the channel contains sufficient tokens. This model assures *execution determinism* (the data produced by a KPN are a function of entries of the KPN) and *monotonicity* (a KPN needs only partial information of the input stream to produce partial information of the output stream). The last property allows *parallelism*, because a process can start the computation of output events without needing the whole input. Testing an input channel for the existence of tokens without consuming them is forbidden. Since there is a total order of events inside a process but no order relation between events in different processes, KPN are only partially ordered. Another view of KPNs, as pointed out by [132], is a set of Turing machines connected by one-way tapes, in which each machine operates on its own working tape. Unfortunately, as a consequence related to the "halting problem"[3], the questions for important properties such termination and memory boundedness are undecidable.

1.3.2.2 Synchronous Dataflow model

One solution consists in using restrictions of KPN where these questions are decidable. *Synchronous dataflow* (SDF) [98] model imposes that a process consumes and produces a fixed quantity of tokens for each firing. All the agents are static and their firing rules do not change during the execution. The model is not synchronous in the same sense as for synchronous languages[4], the term "synchronous" referring to producing and consuming a fixed number of tokens, specified a priori, for each

3. The halting problem states that, given a finite-length program that runs on a Turing machine, it is undecidable to decide always in finite time whether or not the program will terminate.

4. Unlike "synchronous" reactive languages for dataflow dominated systems, like Signal or Lustre, there is no notion of clocks.

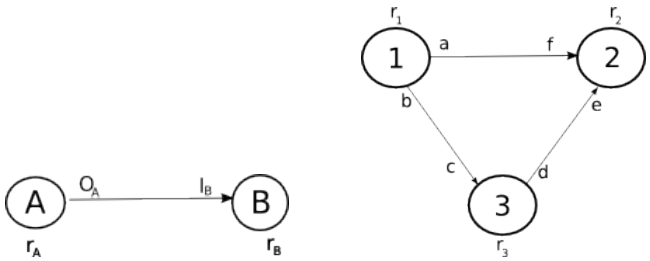
actor. More, production and consumption rates on all arcs are related through *balance equations*.

Suppose that an actor A produces O_A tokens on an output connected to actor B (cf. Fig.1.5-a) which requires I_B tokens on its input in order to execute. Suppose also that the actor A fires r_A times and actor B has a frequency rate of r_B. According to the balance principle, the following equation can be written:

$$r_A * O_A = r_B * I_B.$$

This type of equation can be expressed for each arc in the SDF, forming a system of balance equations. For the simple graph in Fig.1.5-b, the equations are:

$$\begin{aligned} r_1 * a &= r_2 * f \\ r_1 * b &= r_3 * c \\ r_3 * d &= r_2 * e. \end{aligned}$$



(a) Balancing principle

(b) Example of a SDF graph

Figure 1.5: SDF model

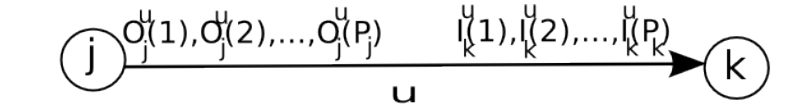
This system is then solved by the compiler to find the vector $r = [r_1, r_2, r_3]$ of firings. As shown in [97], for a connected dataflow, if there is a non-trivial solution for the balance equations, then the solution is unique and it is the smallest positive integer. If there is no solution, the graph has inconsistent execution rates and there is no bounded memory infinite execution of the dataflow. Given a solution, a partial ordering constraints between firings can be specified and a precedence graph can be constructed.

Thanks to the balance principle, the questions of *bounded memory* and *deadlock* are decidable. That and other properties such as determinism and static scheduling at compile time make the SDF a reliable and popular model at the basis of (many) embedded software such as Ptolemy [26], COSSAP [96], System Canvas and DSP Canvas from Angeles Design Systems [124] or Cocentric System Studio from Synopsys.

Even if the SDF model is convenient for certain applications, it comes with a relatively high price: because the number of tokens produced and consumed is fixed, they cannot depend on the data and the application cannot use conditional variations in the flow.

1.3.2.3 Cyclo-Static Dataflow Model

A more flexible extension of SDF is the Cyclo-Static Dataflow (CSDF) model [19] which permits that the number of tokens produced and consumed vary from one activation to another in a cyclic manner. As shown in Fig.1.6-a, every agent j has a sequence of firings $[r_j(1), r_j(2), \ldots, r_j(P_j)]$ of length P_j, the production on the output channel u is defined as a sequence of constant integers $\left[O_j^u(1), O_j^u(2), ..., O_j^u(P_j)\right]$ and the consumption on edge u is defined as a sequence $\left[I_j^u(1), I_j^u(2), ..., I_j^u(P_j)\right]$. The n-th execution of the task j corresponding to the code of the function $r_j((n-1) \bmod P_j + 1$ produces $O_j^u((n-1) \bmod P_j + 1)$ tokens on edge u. A cyclostatic actor j has a firing rule evaluated as "true" for its nth firing if and only if all the input channels contain at least $I_j^u((n-1) \bmod P_j + 1)$ tokens. For an example of a CSDF graph, refer to Fig. 1.6-b.



(a) Balancing principle

(b) Example of a SDF graph

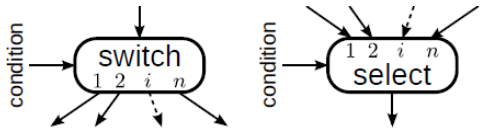
Figure 1.6: CSDF model

It should also be noted that the CSDF model shouldn't be confused with CDFG (Control Data Flow Graph) model [135]. The CDFG are representing the control dependencies between nodes connected through data or control arcs. In a CDFG, each task is executed only once; instead, for a CSFF, there is a continuous data exchange (we can imagine a loop surrounding all the CSDF graph).

1.3.2.4 Dynamic Dataflow Model

A wider dataflow model in which data control execution is allowed is the *dynamic dataflow model* (DDF) [125]. By extending the balance equations, the number of tokens produced or consumed may vary dynamically and thus, the consumption and production cannot be known at compile time. Many of existing dynamic dataflow languages are derived from static dataflow model, by including a limited set of dynamic actors, whose behavior depends on the data.

One of the dynamic dataflow models is the *boolean dataflow*, obtained by the addition of two dynamic actors: "select" and "switch" (see Fig. 1.7) to the synchronous dataflow model. One of the main advantages of this model, as shown by Buck [27], is its Turing completeness. This means that with this programming model, one can implement an universal Turing machine. However, even if it is more flexible and sometimes, it is possible to make approximate scheduling analysis, this model does not allow to answer critical questions like deadlock freeness or bounded memory.



(a) SWITCH actor. (b) SELECT actor.

Figure 1.7: Dynamic actors [33]

1.3.3 Stream programming

Based on dataflow models, stream programming seems a more appropriate framework to express and describe massive parallelism than imperative languages. By their theoretical background, stream languages can guarantee important application properties such as: functional determinism, memory bounded execution or absence of race conditions. Stream programming provides a good choice for applications intended for manycore systems.

A first language to take into account dataflows was Lucid [4] in which even if the program description is sequential, the program can be represented as a KPN. Nowadays, due to the new manycore architecture, there is a regain in interest in steam programming and numerous languages have been proposed. Between the most successful, we can mention StreamIt [2], StreamC [91], Cg [116], Brook [25], ArrayOL [23] or the more recent ΣC [74]. Even if all these programs are based on dataflow models, they are different in the format and type of the streams, the structure of the underlying graph, the expression of data reorganisation and the type of production/consumption (static or dynamic). For a more detailed classification of main stream languages, we refer the reader to [49].

In the sequel, we are introducing more in detail the ΣC programming model and language for embedded manycore and give an overview of the compilation process of such a stream program (see [5], [74] for details).

1.4 Sigma-C programming model and compilation

1.4.1 Sigma-C programming language

ΣC is a dataflow programming model and language designed for parallel programming of high performance embedded manycores processors and computing grids. ΣC model, based on process networks with process behavior specifications, has slightly more expressive power than SDF and CSDFs while being less general than BDF. As such, it has sufficient expressive power for most applications, while allowing to perform a formal analysis for verifying properties like absence of deadlock or memory bounded execution. As a programming language, ΣC relates to StreamIt [2] or Brook [25], and more, it is designed as an extension of the C language, with keywords to define and connect *agents* (individual tasks in the stream model). In addition to all the aspects offered by dataflow programing, ΣC has the ability to specify the productions and consumptions of each task, which information can be

used later in the compilation phase for checking.

An application in ΣC is described as a static instantiation graph of interconnected agents, which does not evolve through time (remains the same during the execution, without any agent creation or destruction or change in the topology). An *agent* is the basic unit of this programming model, behaving as a cyclic machine with variable amount of data. At each transition of the agent corresponds a either fixed or variable number of data produced and consumed on its ports. Except the *user agents*, specific to each application, there are several *system agents* for data distribution and synchronization: *Split*, *Join* (both for distribution of data in a round-robin fashion), *Dup* (for data duplication), *Select*, *Merge*, *Sink*. Similar to agents, *subgraphs* implement only composition and consist of links and data distribution network. For a detailed specification of ΣC language, we refer the reader to [74].

The main characteristics of ΣC will be illustrated through an example, consisting in a moving targets tracking application, a real case study used across this document for an experimental validation of our optimization algorithms.

1.4.2 Example of a ΣC application: Motion detection

The motion detection example represents a video processing application, performing a target tracking on a sequence of related input video frames. Sequential video frames are analyzed and the movement of targets between the frames is outputted. The main idea is to look for temporal redundancy between successive images, using a Block Matching Algorithm (BMA). The current frame from the video sequence is divided into horizontal strips for which we measure the absolute difference between each pixel and the corresponding pixel from the previous frame. For improving the algorithm, a median filter is then applied with the aim of smoothening the strips. For each horizontal strip s, the standard deviations of the absolute difference are computed by macro-blocks and the minimum is selected according to the equation:

$$\sigma_s = \min \sqrt{\frac{\sum_{i=1}^{N}(x_i - m)^2}{N}}$$

where N is the size of the macro-block (given as an input parameter), x_i is the absolute difference in intensity for pixel i and m is the average over x_i. Afterwards, a corresponding binary image B is constructed, by applying to the image a filter, such as, for each pixel i:

$$B_i = \begin{cases} 1 & \text{if } x_i - m \geq \min_s(\sigma_s) \\ 0 & \text{otherwise.} \end{cases}$$

Using this binary image, the connected components are computed for each strip, where there is a difference in pixels. At the end, the connected components for different strips which are overlapping or have common edges are merged. Finally, in the output image, the remaining connected components identifying the moving targets are represented with bounding boxes.

Figure 1.8 [144] gives an overview of the main components of the ΣC dataflow graph for the motion target application. The network is described from left to right. The tasks *io* belonging to a special class of ΣC agents provide a way to handle the input/output. The left ones are in charge of reading two consecutive frames, of height H and width W, and the right one is displaying the current image with the targets identified by surrounding rectangles. The next tasks s are data distribution agents of type *Split* which take as input the current, and respectively, the previous image, and divide them into NB_S strips (in a round robin fashion). Production of these system agents is given by the NB_S parameter and the application size directly depends on this level of granularity, the further treatments being realized for each strip. As such, the following tasks Δ are computing the absolute difference for pixels by strips which is then used by σ to compute the standard deviation by macro-block and select the minimum for each strip. The outputs of Δ agents serve also as inputs for the t agents to construct a binary version of each strip which is further employed by the c tasks to detect connected components by strip. First m vertex representing a *subgraph* (including a *join* system agent and a user agent defining a median filter) compute the minimum deviation between all strips and broadcast the result to all strips. The second m vertex is another subgraph for merging together the bounding boxes found for each strip. The empty vertex is the *Dup* agent, used to duplicate data over all output channels.

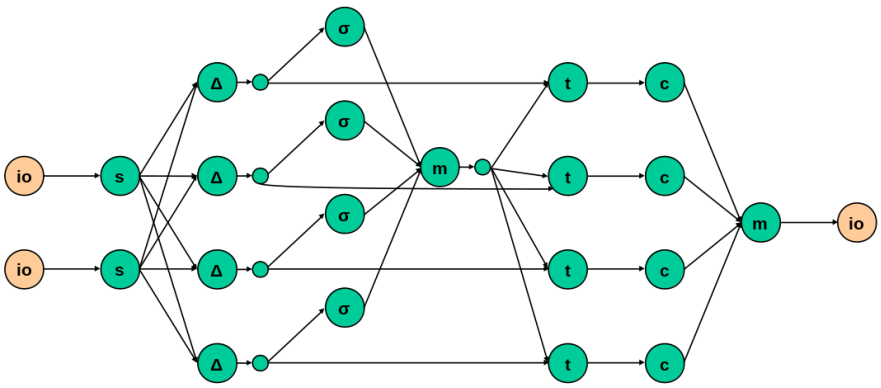


Figure 1.8: A ΣC graph example [144]

1.4.3 Overview of the compilation process

Designing and implementing a parallel application using a dataflow programming language, as the one defined in previous sections, is not enough for running it on an embedded system. The dataflow computation only specifies the application constraints and we need an *execution model* for a formal specification on how to execute the dataflow on an embedded platform. It is the role of the *compiler chain* to connect the dataflow application to the execution model consisting of several steps: parallelism reduction, scheduling, etc. The different steps a compilation process is composed of will be illustrated through ΣC example.

As shown in Fig.1.9 [144], the *compilation process* for ΣC language is organized into four passes.

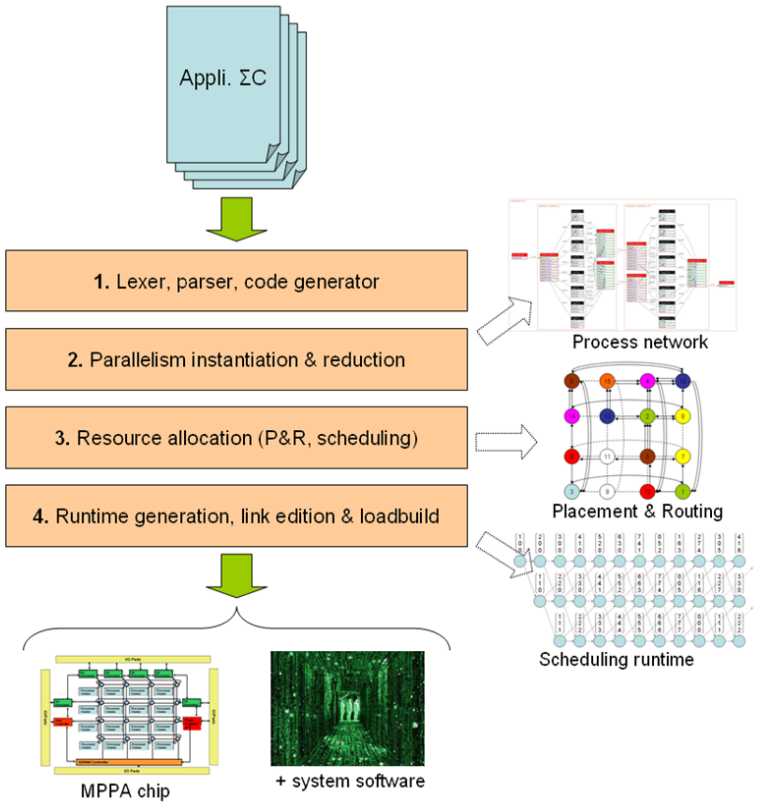


Figure 1.9: ΣC compilation chain [144]

1.4.3.1 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. Afterwards, preliminary C codes are generated from ΣC sources. These codes are intended either for off-line execution (the instantiation codes of the agents), either for further refinement, by substitution (the treatment codes corresponding to treatment agents).

It is important to remark that the code is generated once for all and it is not intended to be modified unless by substitution and only by a C preprocessor.

1.4.3.2 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. The C code, calling adapted APIs, is compiled and executed on the workstation for building the data structure representing the network of processes and generating the data for the agent instances.

This pass also involves a certain number of manipulations on the network of processes. Once the construction of the application graph is complete, parallelism reduction techniques are applied such as the application is rendered compliant with an abstract specification of the system resources.

The parallelism reduction makes use of pattern detection and substitution, by

replacing some parts of the instantiation graph with another subgraph. The modifications on the graph include instance and port creation, deletion or modification and are allowed as long as they do not modify the semantic of the application or the user code. For more details on this compiler extension for dataflow languages, please refer to [45], [34]. The built-ins system agents (split, join, dup etc.) are further combined and transformed into shared memory buffers or NoC transfers. During this stage, it is possible to verify the hierarchical coherence of the agents (for each subgraph verify that its composition implements correctly its state machine) and to perform a safe computation of a dealock-free lowest bound for the buffers sizes of the links (see [145]).

At the end of this pass, it is already possible to make a first execution based on POSIX primitives which allows a functional adjustment of the application code, independent of the architecture.

1.4.3.3 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 (non functional constraints). Next, in order to realize a connection with the execution model, it constructs a folded (and thus, finitely representative) unbounded partial ordering of task occurrences (see [70] for details). This step consists in computing a vector giving, for each agent, the number of occurrences needed for reaching back the initial state. This vector is the basis of an execution cycle which can be repeated infinitely in bounded memory. Then, the vector can be used to realize a first symbolic execution taking into account the partial ordering of tasks occurrences and a second execution which adds to the partial ordering the dependences between production and consumption.

This pass is also responsible of placement followed by routing. Informally, the objectives are: grouping together (under capacity constraints for each cluster of the architecture) tasks which communicate the most, mapping these groups of tasks to the clusters (with a distance criterion) and, finally, computing routing paths for the data traversing the NoC. As such, this step implies the resolution of several discrete optimization problems, which form the object of the present study and which will be presented more in detail in the sequel.

In order to reach an appropriate level of performance, the resource allocation can be performed in a feedback-directed fashion. Certain data (e.g. temporal), characteristic to the application, can be obtained through measuring and simulation and reintegrated to the instances of the problems to solve for obtaining results of better quality. Other parameters can also be evaluated through static analysis.

1.4.3.4 Runtime generation and link edition

The last pass, the ΣC back-end, is responsible of generating the final C code and the runtime tables. Based on the partial orderings from the third pass, the runtime tables make the link with the execution model, by setting parameters of the system

such as the configuration parameters of the NoC or the data structures describing the inter-task communication buffers. Also, during this stage and using C back-up compiler tools, are realized the link edition and the loadbuild.

1.5 Research Motivations

1.5.1 NP-difficult optimization problems related to many-cores

As seen in the previous sections, the compilation process of an application for a massively parallel architecture is becoming rather complex and requires solving a number of difficult and large-size optimization problems. Nowadays, the compiler design implies the application of operations research techniques not only to the so-called back-end (by solving optimization problems such as buffer sizing and instruction scheduling e.g. [76], [20], [101]) but also more upward and all the long of the compilation process, in order to efficiently allocate and exploit the inter-related resources offered by parallel architectures. Between the more recent optimization problems, we can mention the placement/routing for multi-cores or the construction of a partial order under throughput constraints application (e.g. [71], [70]).

Most of existing studies treating optimization problems for embedded parallel architectures propose deterministic models. Still, one of characteristics of these systems is the presence of intrinsic uncertain data occurring in the definition of these problems, such as execution times or latencies. Moreover, experimental studies from both fields of operations research and program compilation (for details, see sections 1.5.3 and respectively 2.1.1) 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. However, we notice only a few studies which take into consideration parameter variations and apply the techniques of optimization under uncertainty to the embedded domain (e.g.[35],[100],[106]).

In order to conceive and develop methods of optimization under uncertainty which are adequate to the domain of compilation for manycores, one should first be able to identify, analyze and, if possible, model the sources of uncertainty specific to this area. As such, one of the research questions to which we are trying to provide elements of response in our study is related to the uncertainty sources and we proceed in the next section with a qualitative analysis of these ones, with a particular emphasis on the execution times.

Also, when designing or implementing an optimization algorithm, one has to compute the computational complexity of the examined problem, in general in function of the size of input data. Since another important characteristic of the optimization problems related to compilation for manycores is their large size and, as we show in the next chapter, dealing with uncertainty for the input parameters increases the complexity, *the approximate algorithms* seem a more appropriate choice

to tackle these problems.

Another important issue we must take into account when conceiving optimization methods for dimensioning embedded systems 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 go along with a worst-case approach. Even if it leads to an over-sizing of the systems, worst-case approach is favored since missing any deadline is highly risky and unacceptable. Our methodology is more oriented towards 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. 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, we admit acceptable deviations and in consequence, avoid oversizing (expensive in terms of hardware) or undersizing (expensive in terms of reliability). Moreover, for a system already dimensioned, we could estimate the level of robustness and specify deviation scenarios for which the system is no feasible or scenarios which could be acceptable.

Furthermore, we are projecting our proposed methodology in the framework of *iterative compilation* and we consider that a first *validation* of the embedded application was realized *a priori*, through for example a simulation or a first execution on the target architecture. The analytical technique most often used in practice for validation of an embedded system remains testing, with the scope of checking for the existence of certain properties or qualities. A systematic procedure for testing comprises, beside test execution and evaluation, test case determination, test data selection and expected results prediction. The determination of test cases consists in defining a certain set of input situations to be analyzed while expected results prediction is in charge of determining the anticipated results and program behavior for each executed test. During test evaluation, actual and expected output values along with actual and expected program behavior are compared for establishing the test results. Therefore, we have at our disposal representative experimental temporal data, obtained during validation, which can be used directly in our optimization method or can be exploited for a start to estimate a probability model. Other available data (for example, the one established by the validation of the embedded system during the phase of expected results prediction) could be then employed for validation of our model (if used) and our optimization approach.

The next section gives a first insight on the relevance of using a probability model when treating uncertainty within an optimization resolution technique.

1.5.2 On the role of a model in optimization under uncertainty

One of the key aspects in the definition of a stochastic optimization problem is the way in which uncertainty is formalized. Since one of the main difficulties of optimizing under uncertainty lies in the computation of the probability distribution, it seems appealing to make use of probability models and analytically compute the distribution. However, a question one should answer before integrating a probability

model into a resolution technique for optimization under uncertainty is "Does the model really help?". As we will see, this question is quite often forgotten, current researches taking a leap of faith when making use of a model.

Let us recall some basic notions about a probability model.

According to textbook definitions [44], a *statistical model* consists of a set of probability distributions on a sample space. A statistical model gives a theoretical distribution which represents the ideal case for an infinite set of observations and describes how the random variables are related to each other. Since the number of observations is limited, it is impossible in general to specify the model precisely and a statistical model contains several repartition functions of the observed sample. Thus, modeling data means specifying the possible distributions of the random variables.

The three basic steps for building a model, independently of the chosen modeling method, are: *model selection*, *model fitting* and *model validation*. They have to be used iteratively until an appropriate model for the data has been found. In order to establish the form of the model to be fit on the data, the first step makes use of plots of the data, some knowledge about the process or phenomenon we want to model and assumptions about data sampling (e.g. normal distributed random errors). Once the basic form of the model has been chosen, another phase is the estimation of the unknown parameters of the function. This is equivalent to solving an optimization problem in which the objective is to produce estimates close to the true value of the parameters. Major methods of parameter estimation are the maximum likelihood and least squares. Model validation is the most important and unfortunately most often outlooked step, in which the model is analyzed to check if the underlying assumptions of the previous steps seem valid.

Most of the approaches for optimization under uncertainty make assumptions about the underlying probability model or use simulations without making a true connection and without a thorough validation with the experimental data. Instead, one of the central ideas of the methodology we propose (see chapter 2), using an oracle for computing the probability and for deciding of the feasibility of a solution, is to take advantage and to rationally exploit the observations of the random data. As such, we suppose a pre-treatment step, in which the observations are analyzed and, in function of the complexity of the modeling process, further behavior is decided. The purpose of this pre-treatment is to build a probability model and estimate its parameters without forgetting to validate it, according to the steps described previously. If the estimated distribution associated to the model is too complicated to be analytically described or computed, and thus it is necessary to employ simulation methods to approximate it, it is better for the overall computation time that the oracle directly exploits the sample in deciding the solution feasibility. Instead, if the estimated distribution of the model has a "nice" analytical form which integration demands a computational time clearly inferior to the time the oracle requires for directly exploiting the sample, it is more appropriate for our approximation method to rely on the model.

Before presenting in detail the methodology, let us now return to the uncertainty data of our context application, in particular the execution times.

1.5.3 Characterization of the uncertainties in the context of manycores

1.5.3.1 Overview of the execution times

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. 1.10 [158] several relevant properties of the execution time for a task are revealed. The darker upper curve represents the set of all execution time. 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 are the minimal, respectively the maximal observed execution times. 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.

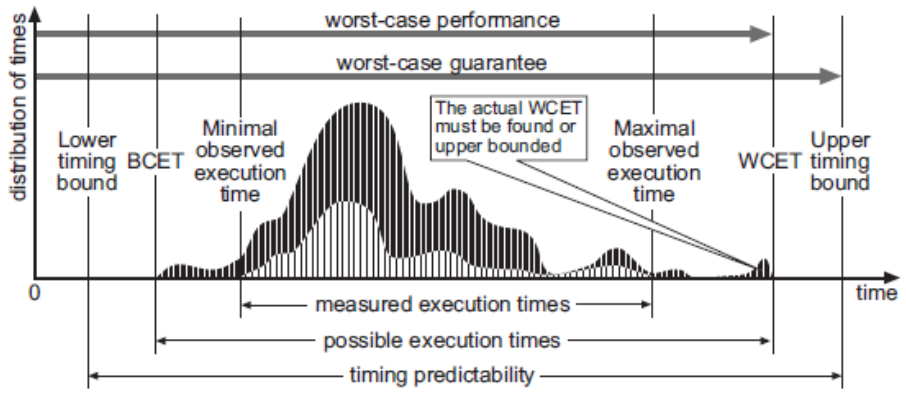


Figure 1.10: Some properties of the execution times of a real-time task [158]

1.5.3.2 Timing-analysis approaches: Deriving and estimating bounds on execution times

Most researches dedicated to the timing analysis of execution times consist in deriving or computing upper bounds for the WCET. The two evaluation criteria for methods or tools for timing analysis are based on *safety* (evaluating if the method or tool produces bounds or estimates) and *precision* (measuring how close to the exact values are those bounds or estimates).

Current approaches for determining bounds or estimates of the execution times are divided into *static methods* and *measurement-based methods*. Methods belonging to first class compute bounds on the execution time, without relying on executing the code on real hardware or on a simulator. They analyze the task code, using some (abstract) models for the hardware architecture, in order to cover the set of all possible control-flow paths and obtain upper bounds. The disadvantage is that they rely on the specification of a processor model and behavior resulting in imprecise results and often overestimated bounds. However, static analysis can be realized without running the program and as such, it avoids implementing complex simulations of the target system. Another important advantage of static methods is their *safety*, the bounds they produced being safe and guaranteed to be always superior to the execution time. As such, they are used for safe scheduling analysis in hard real-time systems. Such static approaches are described in [42], [158]. The second class of methods, based on measurements execute the application or parts of it on the target architecture or a simulator for a given set of inputs. Thus, they can take the measured times and derive or give a distribution of the minimal and maximal observed times but they are not safe since some context-dependent execution paths could be missed. Because they do not need to model the processor behavior, they are simpler to apply to new target architectures and they are able to produce more precise estimates for BCET and respectively WCET, especially for complex processors and applications.

Amongst the issues the above approaches are analyzing we can enumerate: the data-dependent control flows, the context dependence of execution times or the timing anomalies. The control-flow analysis (CFA) applied to the task's control-flow graph (data structure describing the set of all execution paths) determines information about the possible flow of control through the task (infeasible paths, execution frequencies of the different paths, etc.) and it has been called a high-level analysis. For modern processors exhibiting caches and pipelines, the context independence of individual instructions is no longer true since the execution times of each individual instruction may vary by several orders of magnitude in function of the state of the processor. For a task containing two successive code snippets A and B, the execution time of B depends on the execution state produced by the execution of A. As such, there is a need for a so-called low-level analysis or process-behavior analysis to study the behavior of components such as pipelines [77], memory caches [109], [122] and branch prediction [33]. Timing anomalies affect modern powerful processors [111] because of the influence of one instruction on the global execution time of the whole task. One of the anomalies is caused by speculation of the cache on the results of conditional branches. Another type of timing anomalies

are scheduling-caused and usually occur when a set of instructions, with potential dependencies between them, can be scheduled differently on the pipeline units or other hardware resources. In function on the chosen scheduling, executing the instruction or pipelining takes different times.

For a detailed overview and survey of methods and tools for estimating WCET, we refer the reader to [158].

1.5.3.3 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. [28], [54],[112]) 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 [136], analytic profiling [68], [161], [92] and statistic prediction [83], [53].

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 [68], 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 [118] is going further with the analysis, by studying the variations of execution times on multi-core architectures. The experimental work is conducted on samples from two benchmarks SPEC CPU, large sequential applications and SPEC OMP2001 benchmarks, OpenMP applications, by executing each program 30 times on an 8 cores Linux machine with the same train 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 multi-threads), 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.

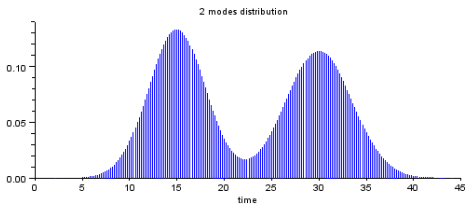
1.5.3.4 Execution times: a qualitative analysis and basic examples

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

Let us give some simple examples. For example, for the computing kernel in Table 1.1, there are two modes for the executions times, possible with different variances, corresponding to the two sequences of instructions. Instead, for the

Table 1.1: A code snippet with a 2 modes distribution

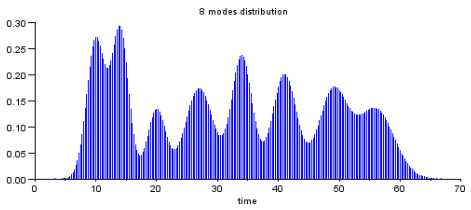
if condition **then**
  S_1
else
  S_2
end if



code in Table 1.2 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 showing a possible envelope of the distribution for the case when $N = 4$). Running a more

Table 1.2: Another code snippet with multi-modal distribution

for $i = 1$ **to** n **do**
  if condition **then**
    S_1
  else
    S_2
  end if
end for



complicated application like X264 encoder clearly shows that the distribution of execution times is difficult to model and that it is multimodal. Fig. 1.11 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

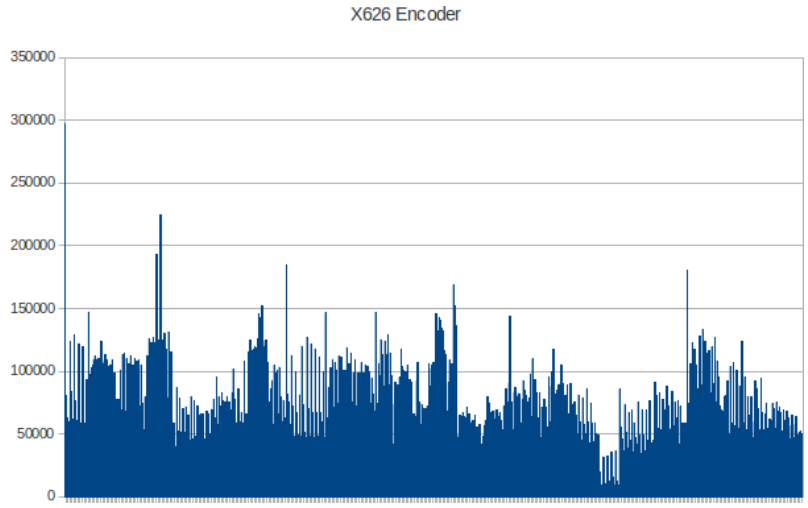
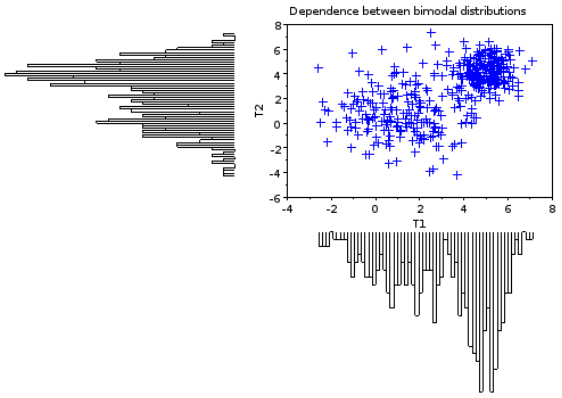


Figure 1.11: Envelope of execution times for frame treatment in a X264 encoder

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 tasks depend, to a certain degree, on the number of effectively treated targets. In Table 1.3, 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, the execution time of task $T1$ is dependent to a certain degree of execution time of task $T2$.

Table 1.3: Code snippet showing possible tasks dependency

T1	T2	Execution times T1 and T2

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

Conclusion

Besides explaining more in details the motivations which conducted our research, this introductory chapter also serves in positioning the context and presenting some fundamental concepts related to manycore systems and dataflow programming (see also [148] for more details on the research motivations).

Also, a qualitative analysis of uncertainty sources for manycore applications is presented and, as the previous section emphasizes, 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.

We do not build further on this assumption for our non parametric robust binomial approach. Since the choice of a probability model seems difficult, the robust binomial approach we propose in chapter 2 is non parametric with almost none or few assumptions on the distributions of the uncertain data. After an introduction in which are presented the existing techniques of solving optimization programs under uncertainty and the difficulties in solving such problems, the next chapter explains in detail the robust binomial approach and possible extensions. The idea of this generic non parametric method is simple and first occurred with the desire to conceive algorithms which match the research context we introduced in this chapter, that is compilation of dataflow applications for manycore systems.

As described in section 1.5, the optimization problems related to compilation for manycore systems are NP-difficult problems, characterized by their large sizes and manipulation of uncertain data, difficult to fully describe but for which we could dispose of experimental samples.

In Chapter 3 and 4, the robust binomial approach was applied in order to solve two of the optimization problems arising in the compilation process: the stochastic partitioning of process networks and the more general problem of placement and routing of process networks (which, for a ΣC, corresponds to the third pass of compilation).

Optimization under uncertainty

Contents

This second chapter presents a general methodology, combining statistical hypothesis testing with heuristic approaches, for solving optimization problems with uncertainty affecting the constraints, under quite mild assumptions on the uncertain data.

A general introduction on the practical relevance of taking into account uncertainty into an optimization problem and different ways of expressing uncertainty are given in section 2.1. Next section focuses on chance constrained programs: after exposing the algorithmic difficulties encountered when facing this kind of problems, a classification of related studies is proposed and each category is presented more in details.

The principles of the non-parametric method we developed, entitled robust binomial approach, are presented in section 2.3. Based on statistical hypothesis tests, the robust binomial method finds approximate solutions to chance-constrained problems (optimization programs with uncertainty affecting the constraints, see details below), guaranteed with a given reliability level $1 - \varepsilon$ and a confidence level of $1 - \alpha$ (both ε, $\alpha \in (0, 1)$). In order to efficiently find *(NS, α)-statistically admissible* solutions (notion defined in section 2.3.4), the robust binomial approach can be combined with existing (meta)heuristic, by modifying the oracle deciding the feasibility of a possible solution. The general methodology of adapting an existing algorithm in order to solve the stochastic version of a problem as well as an example for bin packing are presented in section 2.3.5. Possible extensions for the problems with more initial probability levels or random data in the objective functions are given in the last section.

2.1 What is it meant by "optimization under uncertainty"?

2.1.1 From a deterministic problem to its stochastic counterpart...

A large majority of algorithms and methods conceived for solving combinatorial optimization problems suppose that input data are known precisely. As such, a generic way of mathematically representing an optimization problem is as follows:

$$\begin{aligned} \min_{x} \quad & g(x) \\ \text{s.t.} \quad & G_i(x, \xi_i) \leq 0 \, , i \in \{1, \ldots, m\} \end{aligned} \tag{2.1}$$

where $x \in \mathbb{R}^n$ is the design parameter, $g(x) \in \mathbb{R}$ is the objective function and we have m inequality constraints $G(x, \xi) \in \mathbb{R}$ with $\xi = (\xi_1, \ldots, \xi_m)$ a m-dimensional parameter vector.

However, for real-world optimization problems, one might ask if the formulation above is as general and practical as it seems since the design space is often characterized by data which are uncertain or inexact.

Beginning with the seminal works of Dantzig [46], Charnes and Cooper [38], Miller and Wagner [120], Bellman and Zadeh [6], optimization under uncertainty remains one of the most active domains of research, both in theory and algorithms, and thanks to recent studies, there is an increased regain of interest. The recent case study of Ben-Tal and Nemirovski [10] on a collection of 90 problems from NETLIB library showed that systems optimized in the classical sense (see formulation 2.1) can be very sensitive to small changes and that only 1% perturbation of the data can severely affect the feasibility properties of deterministic solutions.

One such example from [10] is an antenna design problem in which only 5% errors can entirely destroy the radiation characteristics established during nominal optimization. Another example analyzed in [10] is a LP program PILOT4 from Netlib library with 1000 variables and 410 constraints, constraint j being:

$$\begin{aligned} [A_j]^T x \equiv & -15.79081 x_{826} - 8.598819 x_{827} - 1.88789 x_{828} - 1.362414 x_{829} \\ & -1.526049 x_{830} - 0.031883 x_{849} - 28.725555 x_{850} - 10.792065 x_{851} \\ & -0.19004 x_{852} - 2.757176 x_{853} - 12.290832 x_{854} + 717.562256 x_{855} \\ & -0.057865 x_{856} - 3.785417 x_{857} - 78.30661 x_{858} - 122.163055 x_{859} \\ & -6.46609 x_{860} - 0.48371 x_{861} - 0.615264 x_{862} - 1.353783 x_{863} \\ & -84.644257 x_{864} - 122.459045 x_{865} - 43.15593 x_{866} - 1.712592 x_{870} \\ & -0.401597 x_{871} + x_{880} - 0.946049 x_{898} - 0.946049 x_{916} \\ & \geq b \equiv 23.387405 \end{aligned}$$

with $A_j \in \mathbb{R}^n$ the line j of the constraints matrix and $x \in \mathbb{R}^n$. This kind of "ugly" coefficients could model certain technological processes and we could make the hypothesis that they cannot be specified with high accuracy and thus, they are uncertain and have inaccurate last digits. For the optimal solution x^* when the

uncertain coefficients are perturbed within 0.01% margin by independent random perturbations, distributed uniformly, the constraint is violated by at most 150% of the right hand side with a probability of 0.18. In the worst case (all uncertain coefficients are perturbed with 0.01%), the constraint is violated in x^* by 450% of the right hand side.

Let us give another simple example from [80] illustrating that the optimal solution of problem 2.1 might actually be unfeasible if uncertainty on the parameter vector ξ is ignored. Let $\xi = (\xi_1, \ldots, \xi_m)$ with $\xi_1, \ldots, \xi_m$, m independent observations of a standard normal distribution, $x \in \mathbb{R}$, $g(x) = x$ and $G_i(x, \xi) = \xi_i - x$, for all $i = 1, \ldots, m$. If the uncertainty on the parameter is ignored and ξ is substituted by the expected value $\mathbb{E}(\xi)$ in problem 2.1, then the optimal solution is obtained for $x^* = 0$. However, the probability that $x^* = 0$ is a feasible solution equals to

$$\mathbb{P}\{G_i(x^*, \xi_i) \leq 0 \ , \forall i \in \{1, \ldots, m\}\} = \mathbb{P}\{x^* \geq \xi_i \ , \forall i \in \{1, \ldots, m\}\} = 0.5^m$$

As the value of m increases, this probability becomes very weak (e.g. for $m = 7$, it is less than 0.01).

As shown by the previous case studies, taking into account uncertainty affecting the parameters required for optimization is necessary in order to find optimal solutions which are feasible in a meaningful sense. Nevertheless, as we will point out in the next section, optimizing under uncertainty induces several supplementary difficulties and a crucial point is the way uncertainty is formalized and the underlying assumptions.

2.1.2 Uncertainty setting

When formulating an optimization problem under uncertainty two aspects need to be defined: the way uncertainty is expressed and the dynamicity of the problem (or the time when uncertain data is revealed with respect to the time when decisions are taken). [15] proposes a classification of optimization problems under uncertainty in function of uncertainty and dynamicity, shown in Figure 2.1.

The classical Deterministic Combinatorial Optimization Problems (DCOP) correspond to the case of perfect knowledge about the data and, since we are supposing all information is known at decision stage, they are static models.

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 *multi-stage recourse* problems. For both static and dynamic models, there are decisions and 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.

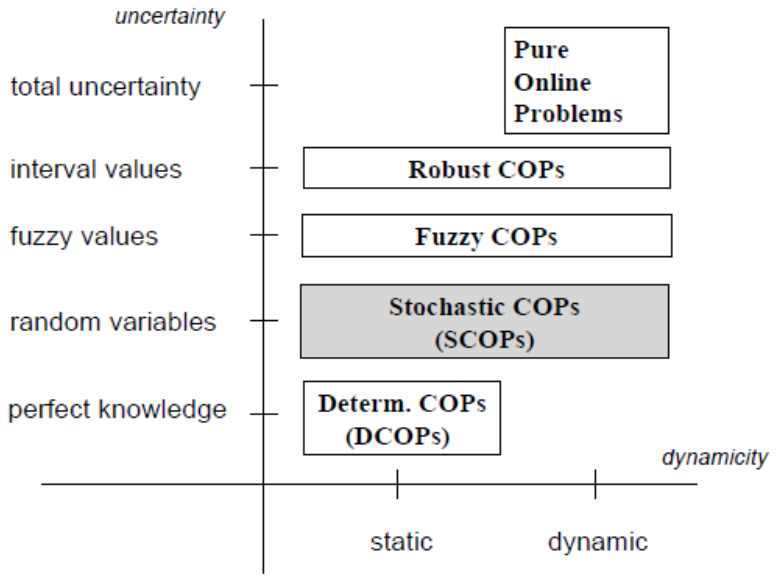


Figure 2.1: Conceptual classification of combinatorial optimization problems (COP) [15]

Another way of representing uncertainty in optimization problems, although minor in the related literature, is by fuzzy quantities for random parameters and by fuzzy sets for constraints to which, instead of probabilities, there are associated *possibilities*. For more details about fuzzy approaches, we refer the interested reader to [6] and [104].

Robust optimization does not need any knowledge about the probability distribution of random data and instead uncertain information is set based. Usually, the decision making process searches for solutions that are feasible for any realization of the uncertainty in the given set. Robust optimization methods are gaining increasing attention lately with recent studies (e.g. [14], [9], etc.) showing the theoretical and practical potential of such approaches as well as making connections with stochastic programming.

The top category from Fig.2.1 corresponds to Pure Online problems, for which the output is produced incrementally, without knowing the complete input and without making any assumption on the new data. The performance is evaluated against the optimal solutions found by an abstract competitor, with a perfect knowledge about past and future data.

Since, in the present manuscrit, we are concentrating on static stochastic programs with uncertainty affecting the constraints, let us provide more background about the general structure, the difficulties in solving this type of problems and the existing resolution techniques.

2.2 Chance constrained programming

2.2.1 Problem statement

The general form of the chance constrained problem we consider here is the following :

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

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 probabilities levels ε_i can be assigned to different constraints:

$$\begin{aligned} \min_{x} \quad & g(x) \\ \text{s.t.} \quad & \mathbb{P}\left(G_i(x,\xi) \leq 0\right) \geq 1-\varepsilon_i \ , \forall i \in 1,\ldots,m \end{aligned} \tag{2.2}$$

The difference between (CCP) and (2.2) formulations is that 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 [134]. 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.

Remark on robust optimization As described briefly previously, robust optimization constitutes an attractive alternative to chance constrained programming for which uncertain parameters are characterized through a set of possible events.

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 of large cost, being guaranteed even for events with a low probability to occur. Recent approaches, 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. For example, Ben-Tal and Nemirovski [9] propose particular relaxations of original optimization problems transforming linear and quadratic

robust programs into deterministic second order cone programs and respectively semi-definite programs, under the assumption that the uncertainty set is ellipsoidal. Bertsimas and Sim [14] model uncertain data as symmetric and bounded random variables and give a linear deterministic counterpart for robust linear problems with J parameters subject to uncertainty in which at most $\Gamma \leq J$ uncertain parameters are allowed to vary. Quite often, even if these transformations provide deterministic convex approximations, the obtained equivalent belongs to a class of complexity more complex than the one of the original program.

Even if robust methods construct solutions which are immune to data uncertainty, in general the quality of the solution is not assessed with probabilistic considerations as in the case of chance constrained programming. However, from our perspective, the probability of constraints to respect a given reliability target is a more intuitive notion, often easier to set for a decision maker.

Moreover, allowing even a very small probability ε for constraint violation can lead to a significant improvement of the optimal solution. The following example, similar to the one from [29], illustrates the fact that, in some situations, the optimal objective obtained using a chance constrained model can be significantly different from the optimum attained by solving the robust approach. Let compare the optimal solution of the robust problem EX_RCP with the one of its stochastic equivalent program EX_CCP from Table 2.1, when the uncertainty set δ is uniformly distributed in $[0, 1]$ and

$$f(x, \delta) = \begin{cases} 10^7 - x & \text{if } \delta \in [0, 10^{-7}] \\ -x & \text{if } \delta \in (10^{-7}, 1] \end{cases}$$

The optimal solution of the EX_RCP formulation is equal to 10^7. Or, when setting

Table 2.1: Example of robust formulation (EX_RCP) against chance constrained program (EX_CCP)

EX_CCP	EX_RCP
$\min\limits_{x \in \mathbb{R}^n} \; x$ s.t. $\mathbb{P}\left(f(x, \delta) > 0\right) \leq \varepsilon$	$\min\limits_{x \in \mathbb{R}^n} \; x$ s.t. $x \in \bigcap\limits_{x \in \mathbb{R}^n, \delta \in \Delta} \{x : f(x, \delta) \leq 0\}$

a probability level $\varepsilon > 10^{-7}$, the probabilistic equivalent EX_CCP will not take into account the violation of constraints for which the uncertainty measure is smaller than ε and thus, the objective found will be equal to zero.

In the following, we will describe only the robust approaches which are accompanied by probabilistic considerations.

2.2.2 Algorithmic challenges

Introduced by the seminal works of Charnes and Cooper [37] and Miller and Wagner [120], chance constrained programs have been extensively studied in the literature of stochastic optimization. Since probabilistic constraints arise naturally

in various contexts, there are a wide range of potential applications for this kind of programs, from engineering design to finance and management. For more details on concrete applications of CCP, please refer to [134].

However, 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 fifty 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:

- Convexity of chance constraints.

For joint chance constrained programs, having this structural property would allow using resolution tools from convex optimization field and thus, finding a global optimal solution. However, the convexity of the feasible set for the probabilistic constraints from CCP program depends not only on the convexity of function G in x but also on the distribution of the random parameter ε. In general, the distribution function of random variables can never be concave or convex. Worse, even if each $G_i(x, \varepsilon)$ is convex, the union of all constraints may not be convex. Let us give a simple example, inspired from [139], showing that even when the functions G_i are linear, the overall program correspond to a non-linear non-convex problem.

$$\begin{array}{ll} \min\limits_{x} & g(x) \\ \text{s.t.} & \mathbb{P}\begin{pmatrix} 2x_1 + x_2 & \geq & b_1 \\ x_1 + x_2 & \geq & b_2 \end{pmatrix} \geq 0.5 \\ & x_1 \geq 0\,, x_2 \geq 0 \end{array}$$

where b_1 and b_2 are two dependent random variable such $\mathbb{P}(b_1 = 5, b_2 = 3) = 0.5$ and $\mathbb{P}(b_1 = 2, b_2 = 4) = 0.5$. Let S_1 be the polyhedral set satisfying $2x_1 + x_2 \geq 5$ and $x_1 + x_2 \geq 3$ and S_2 the polyhedral set satisfying $2x_1 + x_2 \geq 2$ and $x_1 + x_2 \geq 4$. Even if any (x_1, x_2) from S_1 is feasible for the whole problem and again, any point from S_2 is feasible, the complete set of the problem is defined as the union of S_1 and S_2 which turns out to be non convex (as shown in Fig.2.2).

- Evaluation of the probabilistic constraints.

In order to evaluate, for a given x, the probability that $G(x, \xi) \leq 0$, we need to know the probability distribution of the random vector ξ. So, a first problem raises, the one of modeling random data in practical applications when the involved distributions are not always known exactly and have to be estimated from historical data. Another sensible point is the choice of the reliability level ε which is difficult to be established in absence of any knowledge about the underlying probability distribution.

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. As such, the multivariate distribution of ξ, if given, is often approximated by Monte-Carlo simulations or bounding arguments.

Another crucial aspect when designing algorithms for CCP problems is whether the distribution is continuous or discrete and also if the components of the random data are independent. One of the biggest challenges arises for programs in which decision variables and random data are correlated and cannot be decoupled.

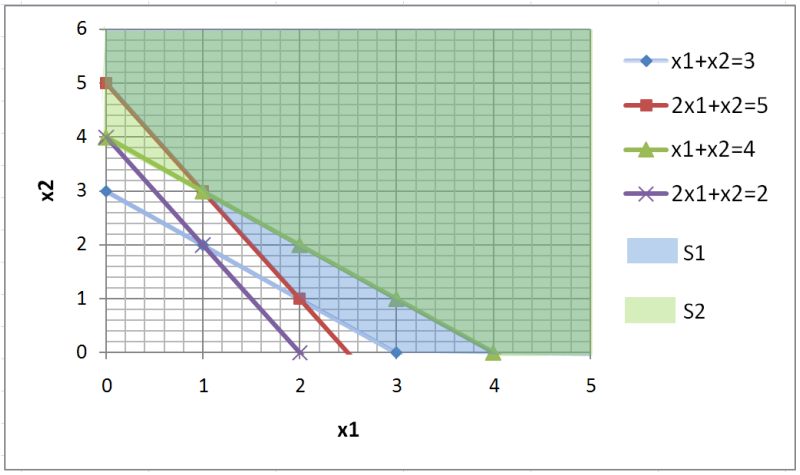


Figure 2.2: Example of a non-convex chance constrained program

Therefore, even for simple cases (e.g. linear programs) the problem (CCP) may be extremely difficult to solve. Table 2.2 shows some of the main theoretical and algorithmic resolution methods proposed for solving joint 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"). Further explanations on each category will be given in the corresponding subsections.

Table 2.2: Methods for solving chance constrained programs

Category	Characteristics	Some references
Convexity studies	Theoretical approaches Particular assumptions on the distribution	[38],[120], [134], [79]
Robust optimization	Relatively simple to apply	[9],[14],[39], [30],[29],[31], [160] [94], [7], [8], [69]
Approximations and sampling	Compute bounds and approximate solutions Usually computationally demanding	[128],[126],[137], [80],[11],[51],[52], [110],[129], [102]
(Meta)Heuristics	Use of precedent techniques for computing distribution	[153],[107],[12],[13],[3]

2.2.3 Convexity studies

Some of the earliest studies from stochastic optimization were interested in establishing conditions in which the probabilistic distributions and the functions defining the constraints define a convex feasible space. As such, almost all exact solutions existing for chance constrained programs require a continuous distribution and a convex structure of the problem.

Charnes and Cooper [38] studied the case of single chance constraints ($m = 1$) when the continuous random variables are only on the right hand-side of the constraints (i.e. completely decoupled of the decision variables) and proposed a deterministic nonlinear equivalent problem. Also, when $m = 1$ and the randomness

is continuous and on the left hand-side, Kataoka [89] proved that these chance constrained programs are convex for independently normal distribution and $\varepsilon \geq 0.5$.

For joint chance constrained programs with more than one constraint, the most difficult case to solve is the one in which the random distributions are affecting the left hand-side. Instead, for random right hand-side parameters, Prékopa [134] showed that if the random variables have a log-concave distribution (e.g. the multivariate normal distribution, uniform distribution are log-concave), then the initial probabilistic program can be rewritten as a convex deterministic equivalent problem. Prékopa also proved that for normal distributed left hand-side parameters, if all covariance and cross-variance matrices for columns or rows are proportional between them, then the problem is convex. A later study of Henrion [79] gives conditions in which program is convex for left hand-side random parameters normally distributed with independent components.

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.

2.2.4 Robust approaches

Some of the recent studies providing less conservative solutions to robust optimization problems [9], [14], [58] have also taken into account probabilistic requirements. Under quite mild assumptions about distributions such as independence of components, known mean symmetry and bounded support, Bertsimas and Sim [14] and Ben-Tal and Nemirovski [9] propose deterministic counterparts and probability bounds against constraint violation for robust linear optimization problems. However, dependence between uncertain parameters is considered in [14] only for a specific model in which, for the constraint matrix of random parameters, only some coefficients from a same row are correlated. The assumption of distributional symmetry is also limiting for many applications, which leads to [39], another study proposing a generalization of the approach of Bertsimas and Sim to asymmetric random variables, using forward and backward deviation measures. Still, this approach is not applicable when we don't know the support, even if the first two moments of the distribution are known. Also, in some situations when the ratio between deviation measures and the standard deviation is large, the approximation found can be too conservative.

Other models, based on the polyhedral properties of the robust problems, with a direct application for the network field, have been proposed in [7] and [94].

Other exact methods, inspired from the scenario approach coming from the robust optimization field, consist in supposing that the probability distribution is discrete, as well as having a bounded support. As such, the probability for each scenario is supposed to be known or computable. Under these assumptions, the problem is formulated as a mixed-integer linear program (MILP) and solved with exact approaches. An example is the model used in [69] for solving the quadratic knapsack problem with probability constraints, solved using semidefinite programming. Although the structure is similar to the one used in our robust binomial approach, this model makes the assumption that the distribution of the

random constraint matrix $m \times n$ is known and has the form $\sum_{A\in\Omega} p_A \delta_A$, with $\sum_{A\in\Omega} p_A = 1$, Ω the event set and δ_ξ the Dirac distribution centered at point $\xi \in \mathbb{R}^{m\times n}$. The equivalent to a linear chance constrained program is the following MILP:

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & Ax \leq b + (1-\chi_A)L,\ A \in \Omega \\ & \sum_{A\in\Omega} p_A \chi_A \geq (1-\varepsilon) \\ & \chi_A \in \{0,1\},\ A \in \Omega. \end{aligned} \tag{2.3}$$

in which $c \in \mathbb{R}^n$ is the cost vector, $x \in \mathbb{R}^n$ is the decision variable vector, $\chi_A \in \mathbb{R}^m$ is a vector of binary variables and L is a suitable large problem-dependent constant. The main drawback of this MILP formulation is that its linear relaxation is often weak.

Other studies (e.g. [30], [29], [160], [31]) were interested in establishing theoretical links between chance constrained programs and equivalent scenario-based formulations (in [29] they are using the notion of sample instead of scenario). Their corresponding key results consist in providing explicit bounds on the number of scenarios or samples required to obtain the required predefined level of probabilistic reliability.

2.2.5 Bounds and approximations

As seen previously, convexity or even quasi-convexity for the probabilistic constraint from problem (CCP) are not always easy to verify and prove. Also, the mixed-integer linear formulation remains interesting to apply when the number of scenarios to take into account is limited. As such, other directions of research consist either in discretization and sampling the distribution or in developing convex approximations.

Between the convex approximations, we can cite the Conditional Value At Risk (CVar) [137], Bernstein approximation [126] or the quadratic approximation of Ben-Tal and Nemirovski [11]. Usually, the proposed approximations find feasible but suboptimal solutions to the original problem without any guarantees on their quality. Furthermore, most of them are applicable only on single chance constraints and so, the joint chance constraints have to be approximated by a set of individual chance constraints, making the solutions for approximations more conservative.[1] Another major drawback of these convex approximations are that they require assumptions on the structure of $G(x,\xi)$ and on the stochastic nature of ε. For example, between the assumptions of Bernstein approximation we can enumerate the independence of the random variables ε_i and the fact that the moment generating function for the distributions are supposed to be efficiently computable. The approximation

1. In order to guarantee the satisfaction of joint constraints, a possible choice consists in using Boole's inequality: $\mathbb{P}\left(\bigcup_i G_i(x,\xi_i) \leq 0\right) \leq \sum_i \mathbb{P}\left(G_i(x,\xi_i) \leq 0\right)$ with $\mathbb{P}\left(G_i(x,\xi_i) \leq 0\right) \geq 1-\varepsilon_i$, $i = 1,\ldots,m$ and $\sum_i \varepsilon_i = \varepsilon$.

programs are commonly solved using non-linear optimization techniques such as the reduced gradient algorithm. If the function approximating the chance constraints is analytically tractable (e.g. for Bernstein approximation), then its evaluation is easy. Otherwise, one must make use of simulation methods such as Monte-Carlo for evaluating these functions (see [137]).

Simulation techniques are also used quite often when a direct evaluation of the feasibility of chance constraints is not possible and the probability has no available closed form. The approximation methods based on sampling are replacing the actual distribution by an empirical distribution estimated by simulation. Originally developed for stochastic programs with objective expressed as an expected value, the *Sample Average Approximations* (SAA) techniques have been applied for chance constrained programs in [110] and [128]. In [110], the sampling method is used to find upper bounds to the chance constrained problem and solving an equivalent mixed-integer formulation of small size. The paper also provides theoretical results on the size of the sample required in order to guarantee a feasible solution with a high probability for initial problems in which the randomness is in the right-hand side. In [128] they are establishing conditions of convergence of a solution to the sample approximation of the original problem in function of the sample size and the probability level. However, the use of Monte-Carlo simulations is too computationally demanding when ε is small and the assumptions made in order to obtain tractable approximations are restricting their applicability to particular cases (e.g. in order to generate Monte Carlo samples, the methods require the full joint distribution).

For solving CCP with random right-hand side, another family of methods which discretize the distribution are based on the notion of p-efficient points [133], [51],[52]. If $F(.)$ is the cumulative distribution of the random parameters ξ and $p \in [0,1]$, a point $v \in \mathbb{Z}^s$ is called a p-efficient point if $F(v) \geq p$ and there is no $y \leq v$, with $y \neq v$ such that $F(y) \geq p$. While earliest study [133] focused in reformulating the CCPs into exact deterministic formulations, the more recent ones [52] are trying to identify useful p-efficient points.

2.2.6 (Meta)heuristics

Actually, since dealing with uncertainty in optimization problems is highly complicated and difficult, the approaches that guarantee to find optimal solutions are more appropriate when solving small size instances and even so, they require a lot of computational effort. In contrast, approaches based on heuristics or metaheuristics are capable of finding good and even optimal solutions to problem instances of realistic size, in a smaller computation time.

However, to the best of our knowledge and as pointed in [15], a survey on existing metaheuristics for dealing with stochastic combinatorial optimization problems, there are only a few heuristics proposed for solving formulation (CCP), corresponding to no-recourse static programs and with uncertainty affecting the constraints.

In [107], the approach consists in using a Monte-Carlo simulation in a genetic algorithm fitness function. For each uncertain parameter, a statistical distribution must be obtained or assumed and the sampling is carried out using either Monte-

Carlo sampling or Latin Hypercube Sampling. If the estimated reliability of meeting one or more constraints is less than the prescribed probability level, the current solution is penalized. As such, the use of sampling is different from our approach and no theoretical guarantees are provided for establishing the number of necessary realizations.

Another method for solving CCPs, suggested in [3], combines a tabu search heuristic with simulation. The evaluation of the feasibility of a solution is realized using two different methods. The first one consists in randomly generating T values for each random variable and computing the average over them in order to evaluate the constraints. The second method uses the central limit theorem to obtain a normal approximation of a sum of independent random variables. Although the first method is sample based, no statistical tools are used in order to determine and reduce T, which is the dimension of the sample employed to estimate the constraint feasibility. Furthermore, the second evaluation makes the simplifying assumption of independence of the random variables.

Another tabu search heuristic is proposed in [153] for solving joint chance constrained stochastic programs with random parameters having discrete distributions. The main focus in [153] is on exploiting the scenario structure: identifying subsets of scenarios that are more important in finding good solutions, adding or removing scenarios at each iteration step. Though the ideas presented are interesting, it seems that the maintenance of the set of scenarios to work with can be quite computationally demanding.

A beam search heuristic, based on the classical Branch and Bound scheme, is suggested in [12] for solving chance constrained programs with integer variables and random right-hand side. In order to evaluate which nodes to explore further, the heuristic is using the lower bound of the optimal solution, computed using the notion of p-efficient point. Since the definition of p-efficient point is using the conditional marginal distribution function, this method supposes as known and computable the distribution of the uncertain variables.

2.3 Robust binomial approach (RBA)

2.3.1 Basic ideas and motivations

Most of the studies mentioned above 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.

We have found that, in many real world situations, the probability distribution is not explicitly known or its integration is too difficult. As illustrated in the previous chapter in section 1.5.3, one such example consists of the execution times of medium-grained computer programs which are random variables difficult to fully describe analytically. However, in practice, we have at our disposal some observations for the uncertain data, obtained, 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 [2].

To the best of our knowledge, the only tractable approximation of the probabilistic constrained programs, which does not impose any restrictions on the structure of the uncertain data [3], is the one derived from the general scenario approach ([30], [31]). The optimization problem (CCP) can then be approximated by the convex program:

$$\begin{aligned} \min_{x} \quad & g\left(x\right) \\ \text{s.t.} \quad & G(x, \tilde{\xi}^{(i)}) \leq 0; \;\; i = 1 \ldots NS \end{aligned} \qquad (RCP_{NS})$$

where $\xi^{(1)}, \ldots, \xi^{(NS)}$ is a sample of size NS of independent and identically distributed observations of ξ and $\tilde{\xi}^{(i)}$ is a realization of $\xi^{(i)}$. Let us recall that ξ is a random vector and that no assumptions are required on its joint probability distribution, in particular with respect to the independence of its components. The scenario approach searches for solutions which satisfy the probabilistic constraints for all the realizations of ξ. The acronym RCP_{NS} refers to the fact that this new formulation is a robust program where, instead of having m constraints, we have $NS \times m$ constraints. As such, the proposed approximation to the original program is often too conservative by finding feasible but suboptimal solutions. Theoretical justification of this approximation scheme can be found in [48], [29].

Our idea is to take advantage of the experimental data and revisit the scenario approach using elementary tools from statistical hypothesis testing theory and directly exploiting the available sample.

Also, in order to face the computational complexity which is one of the major drawbacks of the sample-based method, we propose a general way of integrating it in almost any heuristic algorithm. In this manner, even if the application case requires a high level of precision for the probability constraint threshold ε (which involves the analysis of a large sample), our approximation remains computationally tractable and, as we shall see in the next section, statistically meaningful.

Our algorithm design methodology consists in leveraging existing heuristics for the deterministic case without destructuring them significantly (i.e. at small cost in terms of software engineering) and with an acceptable performance hit.

2. 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 operation.

3. We impose no restriction in particular with respect to random vector component independence

2.3.2 Statistical hypothesis testing

Before presenting the statistical results on which our method is based, let us introduce the following notation:

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.

So the random variable $\chi = \sum_{i=1}^{NS} \chi_i$ follows, *by definition* a Binomial distribution with parameters NS and p_0 ($\chi \sim \mathcal{B}(NS, p_0)$). 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 $1-\varepsilon$:

$$\mathbb{P}(\chi \geq k) \leq \alpha \tag{2.4}$$

For any fixed $p_0 < 1-\varepsilon$, $\mathbb{P}(\chi \geq k)$ is smaller than $\mathbb{P}(\chi' \geq k)$ when $\chi' \sim \mathcal{B}(NS, 1-\varepsilon)$. 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:

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

Short reminder on some general notions. Hypothesis testing is a method consisting in deciding, based on a set of observations, if a default position, called *null hypothesis* (H_0), should be accepted or rejected in favor of the *alternative hypothesis* (H_1). The type I error or the significance level of the test is defined as the risk of rejecting the null hypothesis, when it is in fact true. The roles of the two hypothesis are not symmetrical: the null hypothesis is usually the statement which we do not want to reject if true. Therefore, it is the opposite of what we want to demonstrate: the research hypothesis H_1. As such, we want to minimize the type I error. For example, in the diagnostic check of a disease, we want especially to avoid telling a person is in good health when in reality he is sick: in this situation, the null hypothesis should be chosen that the person is sick [4]. Type II error or the β risk is the probability not to reject H_0 whereas H_1 is true. Usually, type II error is more difficult to estimate. Schema 2.3 represents the four possibilities and the corresponding probabilities of matching and mismatching between the decision believed to be the truth and the "state of nature" which is the actual reality.

4. Example suggested at: http://phdtutor.com/stat_course/Hypothesis_Testing.aspx

Table 2.3: The two types of errors when making decisions using statistical hypotheses

Decision \ State of nature	H_0	H_1
H_0	$1-\alpha$	β
H_1	α	$1-\beta$

In our case, H_0 corresponds to the hypothesis made by caution, which is (intuitively) the hypothesis we wish to reject only if we have statistically significant reasons to do so. We consider that this is the correct setting if we wish to confidently achieve robustness: in a conservative way, by taking small values for α (the type I error), we make no claims of a false null hypothesis without good evidence. Hence, we can conclude, with a high *confidence level* of at least $1-\alpha$, that $p_0 \geq 1-\varepsilon$. For a fixed value of α and ε and giving the sample size NS, we are searching the minimal k for the sample NS such that $\mathbb{P}(\chi > k) \leq \alpha$. The variable χ is the statistic test, following a binomial distribution under the assumption H_0, k is the "critical value" of this statistic test and the set of $\chi > k$ corresponds to W, the one-tailed rejection region of H_0.

In practice, we determined k and, respectively W, the rejection region for a fixed α and $p_0 = 1-\varepsilon$. We were looking for the threshold minimal k such that, if $\mathbb{P}\left(G(x,\xi) \leq 0\right) \leq 1-\varepsilon$, then:

$$\mathbb{P}\{\text{observe k realizations satisfying the constraints}\} \leq \alpha$$

Remark 1. If the two hypothesis were inversed, we will obtain a different statistical hypothesis test:

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

In this case, we will be interested in accepting the null hypothesis and minimizing the probability $1-\alpha'$ that we are wrong (with α' the type I error). As such, for a fixed α', we would be looking for a threshold minimal k' for which, if $\mathbb{P}\left(G(x,\xi) \leq 0\right) \geq 1-\varepsilon$, the following inequality holds:

$$\mathbb{P}\{\text{observe k' realizations satisfying the constraints}\} \leq \alpha'$$

In Fig.2.3, a binomial distribution of parameters $NS = 1000$ and $p_0 = 0.9$ is simulated and for α and α' values fixed to 0.05, the thresholds k and k' were found at 883 and respectively 915. The rejection regions for the two statistical hypothesis tests are pointed in a darker color while the lighter area corresponds to β, the type II error for both tests, more difficult to evaluate.

In the following we are using the first statistical hypothesis test.

Remark 2. For NS large enough, the binomial law $\mathcal{B}(NS,p)$ with $p = 1-\varepsilon$ could be approximated with a normal distribution $\mathcal{N}(NS*p, NS*p*(1-p))$. This approximation [5] can be further improved using a continuity correction.

5. Approximation justified by De Moivre-Laplace theorem (first proved in 1718), showing the convergence in distribution of a binomial law towards a Laplace-Gaussian distribution.

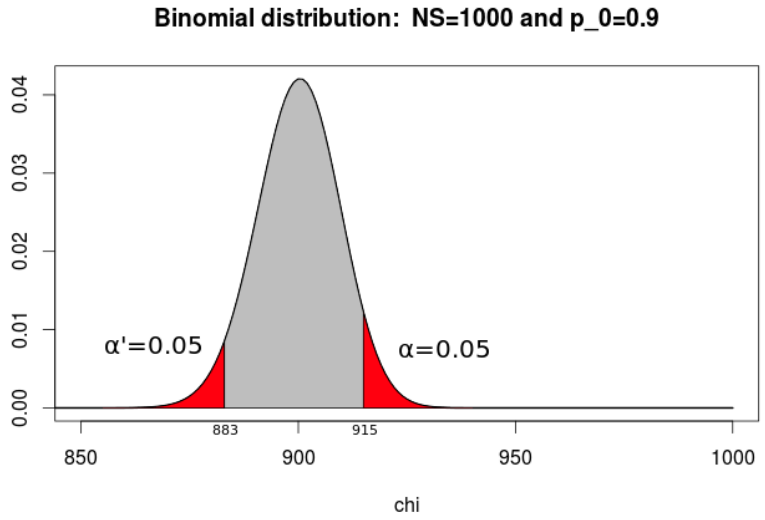


Figure 2.3: Example of test statistics

This basic approximation is not valid for cases when the values for p approaches 0 or 1 (which is often our case) or NS is small. There are various rules of thumb in the literature for specifying the conditions allowing to use this approximation. The most popular rule recommended in statistics texts (without any proof however) states that the normal approximation is adequate as long as $NS * p > 5$ and $NS * (1 - p) > 5$ (or 10 depending on the sources).

Remark 3 An alternative to hypothesis testing is to make use of confidence intervals. Having stated the hypothesis and established α as well as the hypothesized value, a $(1 - \alpha)100\%$ confidence interval (CI) is built such that the null hypothesis is rejected if the hypothesized value does not exist in CI. The above approach is justified by the definition of an unknown parameter θ of a two-sided confidence interval of form $\theta_l \leq \theta \leq \theta_u$ with unreliability α. This entails all the values θ_0 for which the null hypothesis $H_0 : \theta = \theta_0$ would not have been rejected in the observed sample when a two-sided test with α as type I error would have been applied. Any value θ_0 outside the bounds is "improbable'.

In our case, when establishing the number of successes $\chi \geq k$ such that the null hypothesis $H_0 : p_0 = 1 - \varepsilon$ is not rejected (for a significance level α against $H_1 : p_0 > 1 - \varepsilon$), we can also determine an one-sided binomial confidence interval for the proportion of successes k/NS with a given reliability $1 - \alpha$ and $p_0 = 1 - \varepsilon$. There are several ways of computing confidence intervals for a binomial proportion: the "exact" Clopper-Pearson interval [41], the normal approximation interval (also called Wald interval) [156], Agresti and Coull interval [1] etc.

The textbook confidence interval most commonly used for a binomial proportion remains the normal approximation interval. In our case, if $\widehat{p} = \frac{\chi}{NS}$ is the proportion of constraints being respected, the interval we are looking for is $\widehat{p} + z_{1-\alpha}\sqrt{\frac{1}{NS}\widehat{p}(1-\widehat{p})}$ with $z_{1-\alpha}$ the $1-\alpha$ percentile of a standard normal distribution. It is widely recognized that this confidence interval performs poorly especially when the sample proportion p is too near to 0 or 1, in which case the normal approximation is not adequate (see Remark 2).

The "exact" Clopper-Pearson interval is the inversion of the equal-tail binomial test. By guaranteeing that the actual probability is always equal to or above the nominal confidence level, it is however rather conservative. Under the hypotheses stated above, for $NS = 1000$, $p_0 = 0.9$, $\alpha = 0.01$ and $k = 921$ the 99% percent confidence interval using Clopper-Pearson method is $[0.8989388, 1.0000000]$ (R command "binom.test(921,1000,0.9,alternative="g",0.99)"). If we keep the null hypothesis and we state differently the alternative hypothesis: $H_1 : p_0 < 1 - \varepsilon$, the 99% percent Clopper-Pearson confidence interval for $NS = 1000$, $p_0 = 0.9$, $\alpha = 0.01$ and $k = 921$, is $[0.0000000, 0.9396377]$.

For a detailed study comparing the different alternatives for interval estimation of a binomial proportion, we refer the reader to [24]. Along with the risk I error, the confidence interval could be used to measure the probability feasibility of a solution to (CCP) problem.

Let get back now to the robust binomial approach with an analysis on the threshold k.

2.3.3 Sensitivity analysis on the values of parameters for RBA

This subsection tries to determine the influence of parameters ε, α and NS on the threshold k.

The initial reliability level $1-\varepsilon$ from (CCP) is problem specific and thus, can be more or less high depending on the guarantee the decision maker wants to have on the constraints. The level of $\alpha \in (0, 1)$ for a statistical test is commonly set between 1% and 10% and again depends on how much risk the decision maker is willing to take.

Table 2.4 shows some minimal values for k in function of the sample size NS, $\varepsilon = 0.10$ and $\alpha = 0.05$. For example, for establishing that an inequality holds with a preset probability level of $1 - \varepsilon = 0.90$ and with a confidence level $1 - \alpha = 0.95$, 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.

Table 2.5 gives a deeper insight about the minimal number of constraints to respect depending on ε, the prescribed probability level and α, the confidence level when NS, the size of the sample, is equal to 100, 1000 and 10000. It should be remarked that for respecting higher probability and confidence levels, a more important sample size is needed. However, a sample size of 1000 seems sufficient even when $\varepsilon = 0.01$ and $\alpha = 0.01$. Also, it is possible to obtain the same value of

k for different values of ε and α (for example, for a sample of size 1000, we obtain $k = 948$ for $\varepsilon = 0.07$ and $\alpha = 0.01$ and same value also for $\varepsilon = 0.05$ and $\alpha = 0.6$).

Table 2.4: Examples values for $k(NS, 0.90, 0.05)$ in function of NS.

NS	$k(NS, 0.90, 0.05)$
10	-
20	-
30	29
40	38
50	48
100	94
1000	915

Table 2.5: Values of k in function of α and ε

	NS=100			NS=1000			NS=10000		
α \ ε	0.01	0.05	0.1	0.01	0.05	0.1	0.01	0.05	0.1
0.01	-	99	96	996	965	921	9922	9550	9069
0.05	-	98	94	995	961	915	9916	9536	9049
0.1	-	98	94	994	959	912	9913	9528	9038

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 $\alpha \in]0, 1[$).

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 $NS = 20$, see Table 2.4). 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 ($1 - \varepsilon$). We remark that its computation does not depend on the number of decision variables as in [30], nor on complicated complexity measures from Vapnik-Chervonenkis theory as in [154].

Let us now analyze the effect of varying the confidence level $1-\alpha$ on the threshold k for a fixed probability level $1-\varepsilon$ and a fixed sample size. Fig. 2.4 (a) shows the values of k for different values of ε when the sample size is 1000 and α taking different values in $(0,1)$. It seems that the more risk we are accepting, the more the value of k diminishes (linearly) and so the less is the number of realizations needed for satisfying the constraints. However, for an acceptable risk error α (less than 10%), 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. Fig. 2.4 (b) shows the variations of k in function of $\varepsilon \in (0,1)$ for a fixed confidence level (0.01, 0.03, etc.) when the sample size is 1000. As expected, for an important probability guarantee, the number of realizations satisfying the constraints has to be higher. For a sample of size 1000 and different levels of $1-\alpha$, we remark an augmentation of 85 in average for the value of k when $\varepsilon = 0.01$ than the value of k for $\varepsilon = 0.1$. Same phenomenon happens for a sample of size 10000, when in average we have to have 883 more realizations satisfying the constraints for $\varepsilon = 0.01$ than for $\varepsilon = 0.1$.

2.3.4 Chance constraints and sampling

The statistical theory above can be applied for obtaining a statistically significant approximation model to the initial program (CCP). Let us first define the notion of *(NS,α)-statistically admissible solution* for a chance constrained program.

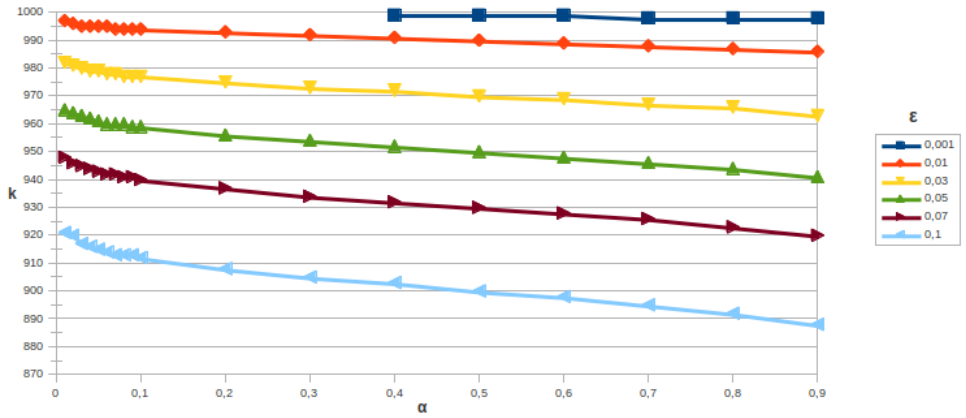
Definition 2.1. *Let $\varepsilon, \alpha \in (0,1)$ and let suppose we are given a sample of size NS for ξ. A solution $x_{(NS,\alpha)} \in \mathbb{R}^n$ is (NS,α)-statistically admissible with a confidence level of $1-\alpha$ for a sample size of NS if, for $p_0 = \mathbb{P}(G(x_{NS,\alpha},\xi) \leq 0)$, $\mathbb{P}(\chi \geq k) \leq \alpha$, with $\chi \sim \mathcal{B}(NS, p_0)$.*

In order to obtain a relevant equivalent program to (CCP) model, we make the following assumptions about the random vector ξ, represented by a sample of size NS of observations $\xi^{(i)}$, with $i = 1, \ldots NS$:

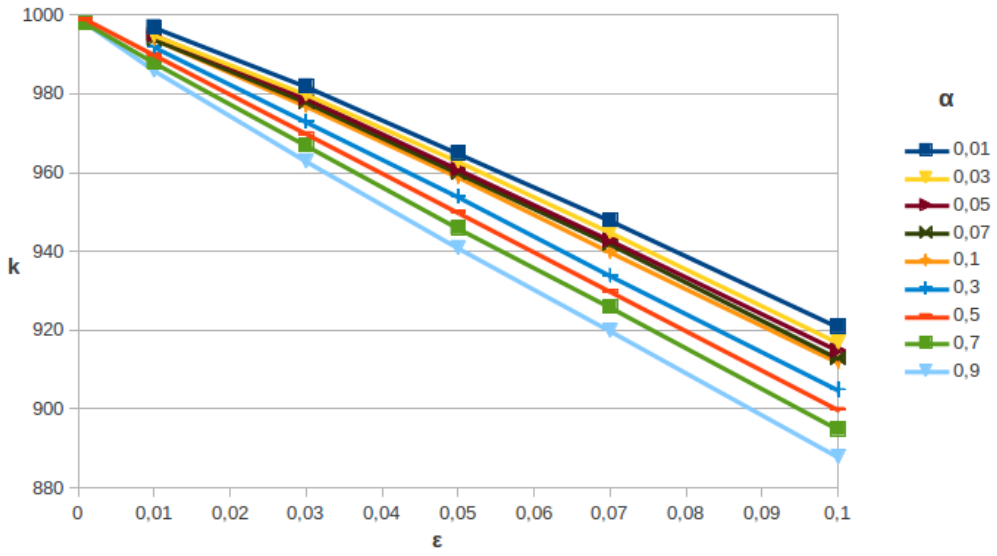
Assumption 2.1. *NS, the size of the sample for the uncertain vector ξ, is finite and sufficiently representative.*

Assumption 2.2. *The sample for ξ is composed of independent and identically distributed (i.i.d.) observations: $\xi^{(1)}, \ldots, \xi^{(NS)}$.*

We would like to attract the attention of the reader on the fact that we are not treating time series. As such, our 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.*



(a)



(b)

Figure 2.4: (a)NS=1000 (b)NS=10000

Moreover, 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 re-sampling, such as bootstrapping [57]. However, it is important that the initial sample is representative of the distribution. We underline that we are not concerned by the acquisition of representative experimental data. This stage has to be realized a priori at system level, for example during the validation stage and needs to be done regardless of the method used for solving the chance constrained program. If we take the case of a video encoder for example, the validation tests should provide representative samples of video sequences which can be used for building our approximation program. Afterwards, in order to validate the robust approach, we need other video samples, statistically identical but, of course, different from the first ones. It should also be remarked that in contrast with other existing methods, our sample acquisition as well as possible treatments (such as parameter estimation or bootstrapping) are made *before* applying the robust binomial approach, intended to find an approximate solution to the (CCP) problem.

Let 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), we can determine $k(NS, 1-\varepsilon, \alpha)$. Therefore, we can use $\tilde{\chi}_i$, the realization of the variables χ_i, and replace the probability constraint

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

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

$$\begin{aligned} \min_x \quad & g(x) && && \text{(RBP)} \\ \text{s.t.} \quad & \sum_{i=1}^{NS} \tilde{\chi}_i \geq k(NS, 1-\varepsilon, \alpha) && && \\ & G(x, \tilde{\xi}^{(i)}) \leq (1 - \tilde{\chi}_i) L; && i = 1, \ldots, NS && (2.5) \\ & \tilde{\chi}_i \in \{0, 1\}; && i = 1, \ldots, NS && \end{aligned}$$

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 2.5 verify the respect of the 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, $L = \sum_{i=1}^{m} w_i$.

Theorem 2.1. *A feasible solution for* (RBP) *problem is a* (NS, α)*-statistically admissible solution.*

The proof is immediate and it is justified by the way (RBP) is formulated. The variables χ_i associated to $\xi^{(i)}$ correspond to Bernoulli independent observations (see

assumption 2.2) and their sum is following a binomial distribution of parameters NS and $p_0 = 1 - \varepsilon$. A feasible solution for (RBP) have to respect the first constraint, which assures that, with a confidence level of $1 - \alpha$, the threshold k is respected.

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 our 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.

Furthermore, we can make use of the existing heuristic algorithms developed for the deterministic version of a problem and extend them to the stochastic case. Having at our disposal a sample verifying assumptions 2.1 and 2.2, 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.

The main steps of an overall optimization process for solving a CCP program integrating an heuristic based on the robust binomial approach are presented in Table 2.6.

Table 2.6: Solving a chance constrained problem with a RBA-based heuristic...

problem formulation: g, G, x, ξ, ε
sample acquisition for ξ
pre-treatment of the sample: preliminary analysis, bootstrapping etc.
check existence of heuristic for solving the deterministic version
if $\exists$heuristic for deterministic case **then**
    choice of the "appropriate" existing heuristic for deterministic case
    replace the admissibility oracle with a stochastic one
    integrate the stochastic oracle to the chosen heuristic
else
    choice of an "appropriate" heuristic
    define a stochastic "admissibility" oracle
    integrate the stochastic oracle to the chosen heuristic
end if
validation of the method

We are not concerned in this chapter by the first two steps: formulating the problem as a (CCP) program and the acquisition of a sample for the uncertainty data ξ, which are taken as granted. Our interest resides in the resolution techniques

and our belief is that solving the problem should be realized by taking into account the specificities of ξ, which justifies step 3.

During this preliminary step, the samples we have at our disposal are analyzed: if a "nice" analytical distribution model can be associated (e.g. Gaussian laws), then we proceed with an estimation method for finding the corresponding parameters. If, on the contrary, data analysis suggests that the distribution underneath is too complicated to be analytically described, we verify that the condition of applicability of the robust binomial approach are verified.

Since the first assumption remains general and must also hold for estimation methods, we only have to make sure we dispose of a sufficient sample. As such, if the theoretical minimal size, computed in function of ε and a chosen confidence level α, is inferior to the size of the current sample, we can resort to re-sampling techniques such as bootstrapping. If there are approximate resolution approaches which have been developed for solving the deterministic version of the same problem (step 6), we choose an "appropriate" (meta)heuristic [6] and adapt it, by replacing the admissibility deterministic oracle with a stochastic one (steps 6-8). The stochastic oracle is either exploiting directly the samples with the robust binomial approach or, if a distribution model is assumed, makes use of the estimated parameters for verifying the probability constraint. If no heuristic conceived for the deterministic case has been found, we have to conceive one and integrate a stochastic admissibility oracle (steps 10-12).

Let us now give more insight about the redesign-for-the-stochastic-case methodology when we have at our disposal (which is often the case) an existing (meta)heuristic solving the deterministic version of the problem.

2.3.5 RBA and adapting (meta)heuristics

The robust binomial approach can be applied without major effort within a greedy method already developed for the deterministic version. Since greedy algorithms provide an easy and quick way of finding good quality solutions, they are often a popular choice for a first optimization of large-sized problems. Table 2.7 shows 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, built in a greedy fashion. While there are residual decisions to be made, an oracle is evaluating them for deciding the admissible decisions. Between

6. The notion of "appropriate" heuristic is subjective and consists usually in finding a satisfactory trade-off between implementation effort, quality of the solution and computational time.

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 algorithms stops. As seen, the only major difference when considering chance constraints is in establishing the set of admissible solutions, **by using a stochastic oracle $\mathcal{O}_s$ instead of the original one $\mathcal{O}$** (line 3). 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 RBA and verifies if a residual decision is (NS, α)-stochastically admissible by comparing 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 2.8).

Table 2.7: General schema for a constructive algorithm

Deterministic	Stochastic
Input: g and G functions, ξ	**Input:** g and G functions
1: $R = \{r :$ residual decisions$\}$	**Input:** $\tilde{\xi}^{(1)}, \ldots, \tilde{\xi}^{(NS)}$, ε, α
2: $S^* = \emptyset$	1: $R = \{r :$ residual decisions$\}$
3: **while** $R \neq \emptyset$ **do**	2: $S^* = \emptyset$
4: $D = \{r \in R : \mathcal{O}(r) = True\}$	3: **while** $R \neq \emptyset$ **do**
5: **if** $D \neq \emptyset$ **then**	4: $D = \{r \in R : \mathcal{O}_s(r) = True\}$
6: $d^* = \underset{d \in D}{\operatorname{argmin}}\ g(S^* \cup \{d\})$	5: **if** $D \neq \emptyset$ **then**
7: $S^* = S^* \cup \{d^*\}$	6: $d^* = \underset{d \in D}{\operatorname{argmin}}\ g(S^* \cup \{d\})$
8: $R = R \setminus \{d^*\}$	7: $S^* = S^* \cup \{d^*\}$
9: **else**	8: $R = R \setminus \{d^*\}$
10: break;	9: **else**
11: **end if**	10: break;
12: **end while**	11: **end if**
Output: S^*	12: **end while**
	Output: S^*

Such a context assures a practical and tractable implementation of our approach even for cases when a very high number of constraints is demanded. These situations can arise when ε is set to be really small (e.g. less than 10^{-5}) and thus, it is required to have a large minimal size of the sample. For example, a problem with probability level $\varepsilon = 10^{-5}$ and, accordingly, a confidence level $\alpha = 10^{-5}$, requires a sample of minimal size 10^6 which, although large, is not prohibitive. Additionally, in order to obtain a more rapid computation, the operation of counting the constraint violations can be parallelized without major effort [7].

7. A one line OpenMP pragma will do the trick.

Table 2.8: Deterministic oracle vs. stochastic oracle

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

Input: $r \in R$, G, ξ
if $G(r, \xi) < 0$ **then**
  return $True$
end if
return $False$
Output: $True$, $False$
	Input: $r \in R$, G, $\tilde{\xi}^{(1)}, \ldots, \tilde{\xi}^{(NS)}$
	Input: NS, ε, α
	Compute $k(NS, \varepsilon, \alpha)$
	$nbRespConstr = 0$
	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 $False$
	Output: $True$, $False$

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. For example, the same methodology could be used to integrate RBA into an existing GRASP (Greedy Randomized Adaptative Search Procedure) algorithm to solve the stochastic version of the problem. GRASP, a multi-start heuristic is composed of a construction phase and an improvement phase.

The first phase consists in building a feasible solution S with a greedy randomized algorithm and its adaptation for the stochastic case could be an algorithm similar to the one shown in Table 2.7.

The second phase takes the solution S and tries to ameliorate it, through a local search procedure, by exploring the neighborhood of S. The only difference between a generic local search method for the deterministic case and its adaptation to the stochastic version consists in deciding which of the neighbors n of S is a possible new admissible solution, based on a deterministic oracle $\mathcal{O}(n)$ or a stochastic oracle $\mathcal{O}_s(n)$ respectively. The structure of the oracles $\mathcal{O}$ and $\mathcal{O}_s$ could be the same as before or they could be implemented more efficiently, using the fact that the neighbors are obtained from a current admissible and respectively (NS, α)-statistically admissible solution.

Let us provide an example of a possible application of the RBA method along with a hill climbing heuristic for the classical problem of *bin packing*. The variant of bin packing we consider here is the following:

Bin packing example. Giving a set of n items, each item i having a certain weight w_i and a value p_i, the objective is to place these items in a fixed number of m bins while respecting the available weight C_j of each bin j in order to maximize the overall value. The problem can then be formulated as follows:

$$\max \quad P = \sum_{i=1}^{n} \sum_{j=1}^{m} p_i x_{ij} \tag{2.6}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} w_i x_{ij} \leq C_j \qquad \forall j = 1, \ldots, m \tag{2.7}$$

$$\sum_{j=1}^{m} x_{ij} = 1 \qquad \forall i = 1, \ldots, n \tag{2.8}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i = 1, \ldots, n;\ \forall j = 1, \ldots, m. \tag{2.9}$$

where x_{ij} are binary variables such that:

$$x_{ij} = \begin{cases} 1 & \text{if item } i \text{ is put in bin } j \\ 0 & \text{otherwise.} \end{cases}$$

Constraints (2.7) verify the respect of the maximal capacity for each bin while constraints (2.8) make sure that the item i has been allocated to a single bin.

Supposing that the weights of the items $w_i \in \mathbb{R}$ are random and we want a probability of validity for the constraints on the bin weights superior to a threshold $1 - \varepsilon$, we obtain the chance constrained version of the bin packing problem. Thus, constraints (2.7) are replaced with:

$$\mathbb{P}\left(\sum_{i=1}^{n} w_i x_{ij} \leq C_j;\ \forall j = 1, \ldots, m \right) \geq 1 - \varepsilon$$

Let us define the notions of *admissible solution* and respectively (NS, α)-*statistically admissible solutions* for the bin packing problem.

Definition 2.2. *A placement of items into the m bins is a deterministic admissible solution to the bin packing problem if it respects constraints* (2.7)-(2.9).

Definition 2.3. *Let $\varepsilon, \alpha \in (0,1)$ and let assume that we dispose of a sample of size NS of i.i.d. realizations $\tilde{w}_i^{(1)}, \ldots, \tilde{w}_i^{(NS)}$ for all weights of the items $i = 1, \ldots, n$. A placement of items into the m bins is a (NS, α)-statistically admissible solution if constraints (2.8)-(2.9) are respected and if:*

$$\sum_{p=1}^{NS} \tilde{\chi}_p \geq k(NS, \varepsilon, \alpha)$$

with k computed as described before and:

$$\tilde{\chi}_p = \begin{cases} 1 & \text{if } \sum_{i=1}^{NS} \tilde{w}_i^{(p)} x_{ij} \leq C_j;\ \forall j = 1, \ldots, m \\ 0 & \text{otherwise.} \end{cases}$$

Table 2.9 gives the main steps of a simple hill climbing heuristic for bin packing for the deterministic and respectively, stochastic case, making use of the notions defined before. The initial solution s_0 could be a random allocation of items to the bins or it could be constructed into a greedy fashion, through First-Fit Decreasing (FFD) or Best-Fit Decreasing (BFD) algorithms.

Table 2.9: General schema for hill climbing heuristic: bin packing problem

Deterministic	Stochastic
Input: n, w_i and p_i for $i = 1, \ldots, n$, m, C_j for $j = 1, \ldots, m$	**Input:** n, p_i for $i = 1, \ldots, n$, m, C_j for $j = 1, \ldots, m$
	Input: $\tilde{w}_i^{(1)}, \ldots, \tilde{w}_i^{(NS)}$, $\forall i = 1, \ldots, n$ $\varepsilon, \alpha \in (0,1)$
Input: s_0, initial solution	**Input:** s_0, initial solution
1: $s' = s_0$	1: $s' = s_0$
2: stop=$False$	2: stop=$False$
3: **repeat**	3: **repeat**
4: $N_a = \{n_a \in N : \mathcal{O}(n_a) = True\}$	4: $N_a = \{n_a \in N : \mathcal{O}_s(n_a) = True\}$
5: bestN = $\underset{n_a \in N_a}{\text{argmax}}\ P(n_a)$	5: bestN = $\underset{n_a \in N_a}{\text{argmax}}\ P(n_a)$
6: **if** $P(\text{bestN}) \geq P(s')$ **then**	6: **if** $P(\text{bestN}) \geq P(s')$ **then**
7: s' = bestN	7: s' = bestN
8: stop=$False$	8: stop=$False$
9: **else**	9: **else**
10: stop=$True$	10: stop=$True$
11: **end if**	11: **end if**
12: **until** stop=$True$	12: **until** stop=$True$
Output: s'	**Output:** s'

A simple neighborhood N of a current solution s' can consist in moving an item into a different bin or exchanging items placed in different bins. Once neighborhood N has been defined, the possible neighbors are analyzed and are kept (in N_a) only when are considered to be an *admissible solution* and respectively a (NS, α)-*statistically admissible solutions* (line 5). The best neighbor *bestN* is chosen and if it improves the total value of the solution, it is kept for the following iteration (lines 6-9).

The only major difference between these two algorithms is in deciding the list N_a of admissible neighbors with the help of a stochastic oracle $\mathcal{O}_s$ instead of the deterministic oracle $\mathcal{O}$. While in the deterministic case, the oracle verifies the respect of the capacity of each bin, the stochastic oracle verifies, for the given sample, that the capacity of each bin is respected with a threshold ε and a confidence level α (see Def. 2.3).

This methodology could be extended to adapt other (meta)heuristics such as simulated annealing, tabu search or genetic algorithms for solving the stochastic version of a problem with the RBA approach. Both based on local search, simulated annealing and tabu search methods are similar to the hill climbing with respect to the exploration of neighborhoods for a current solution. As such, their adaptation to the stochastic case could follow the same approach as in the case of hill climbing heuristic. For the genetic algorithms, one way to solve the stochastic version is to integrate the robust binomial approach into the fitness evaluation function and penalizing those individuals which do not respect the threshold k of minimal constraints to be satisfied.

2.3.6 Generalization of the RBA

2.3.6.1 Chance constrained programs with more than one probability levels

The robust binomial approach can be easily generalized to treat different levels of reliability for chance constrained programs. Actually, the probability constraint from CCP only guarantees that the constraint will be respected with a probability level $1-\varepsilon$. However, when the constraint is not respected with probability ε, it does not provide any information about the degree of violation. Note that knowing the magnitude of the constraint violation can also be important, especially for heavy-tail distributions (although the efficiency of the RBA has not been yet assessed for these type of distributions).

One solution is to enforce different levels of protection by defining alternative thresholds for probabilistic guarantees when the primary target is not achieved. Let suppose that the r probability levels are $\varepsilon_1 = \varepsilon < \varepsilon_2 < \cdots < \varepsilon_r$ and $\forall j = 1 \dots r,\ \varepsilon_j \in (0,1)$. As such, if the probability constraint $\mathbb{P}\left(G(x,\xi) \leq 0\right) \geq 1 - \varepsilon_j$ is not achieved, then the probability target $1 - \varepsilon_{j+1}$ is set instead. This results in a program with a set of at most r independent probability constraints. It should however be noted that the larger r is, the larger is the computational effort to solve this type of problem.

These successive independent chance constrained programs could still be replaced by applying for each of them our robust binomial approach under the assumptions made earlier. Eventually, in order to reduce the complexity burden, we can verify in advance if the constraint $G(x,\xi) \leq 0$ is satisfied or not for realization $\tilde{\xi}^i$ and store the result for exploiting it later when verifying the probability constraint for each threshold $1 - \varepsilon_j$. However, with this approach, the final solutions will hold only for each individual ε_j instead of being guaranteed for different levels of probability.

A more interesting way of enforcing different levels of protection is by considering them simultaneously, from the beginning. Let suppose again that we have r probability levels $\varepsilon_1 > \varepsilon_2 > \cdots > \varepsilon_r$ with $\forall j = 1 \dots r,\ \varepsilon_j \in (0,1)$ and this time, for each of them we have different targets to meet $0 \leq M_1 < M_2 < \cdots < M_r$ for the constraint function $G(x,\xi)$. As such, we are looking for a solution which satisfies each of the r probabilistic constraints :

$$\begin{array}{ll} \min\limits_{x} & g(x) \\ \text{s.t.} & \mathbb{P}\left(G(x,\xi) \leq M_j\right) \geq 1-\varepsilon_j,\ \forall j = 1 \dots r \end{array} \qquad (CCP_r)$$

Example. Let give a short example for better illustrating this possible extension of the RBA. Let suppose that we want to guarantee the satisfaction of the constraints with respect to three targets 0, M_2 and M_3 with three different probability thresholds: 0.75, 0.85 and respectively 0.95. As such, we must solve the following chance constrained program:

$$\begin{array}{ll} \min\limits_{x} & g(x) \\ \text{s.t.} & \mathbb{P}\left(G(x,\xi) \leq 0\right) \geq 0.75 \\ & \mathbb{P}\left(G(x,\xi) \leq M_2\right) \geq 0.85 \\ & \mathbb{P}\left(G(x,\xi) \leq M_3\right) \geq 0.95 \end{array}$$

As such, we are looking for a solution which provides alternative guarantees when the first constraint is not respected, by allowing to achieve M_2 with a probability of at least 85% and M_3 with at least 95% probability.

In order to solve this generalization which assures simultaneously different probability levels, the robust binomial approach has to be applied for each threshold $1-\varepsilon_j$. We obtain the following approximation for (CCP_r) formulation:

$$\begin{array}{lll} \min\limits_{x} & g(x) & \\ \text{s.t.} & \sum\limits_{i=1}^{NS} \tilde{\chi}_{j_i} \geq k_j(NS, 1-\varepsilon_j, \alpha) & j = 1, \dots, r \\ & G(x, \tilde{\xi}^{(i)}) \leq (1-\tilde{\chi}_{j_i})L + M_j; & i = 1, \dots, NS;\ j = 1, \dots, r \\ & \tilde{\chi}_{j_i} \in \{0,1\}; & i = 1, \dots, NS;\ j = 1, \dots, r \end{array}$$

with k_j the number of minimal constraints to get respected for probability level $1-\varepsilon_j$ (we suppose the same confidence level $1-\alpha$ for all but it is not mandatory). $\tilde{\chi}_{j_i}$ corresponds to a binary varible such that, $\forall j = 1, \dots, r$:

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

Again, due to the computational complexity (an increase factor of r compared to the original (RBP) program), it is more appropiate to make use of heuristic approaches and conceive a stochastic oracle for checking feasibility of the solutions, by counting the number of respected constaints.

2.3.6.2 Programs with random objective function

Until now, we have considered chance constrained problems of the form CCP, thus static programs with randomness affecting only the constraints. Although, there can be cases in which the objective function of this problem is also random.

For a static program when only the objective function to minimize $h(x, \nu)$ is random (with ν the random data), there are several ways to handle the uncertainty (see [134] for more details):

1. Converting the problem into a deterministic one by considering the expectation value of the random variable.

2. Use a policy based on the notion of efficient points (as defined previously). Another variant of this principle is to replace the objective function by a linear combination of expectation and standard deviation.

3. Introduce a probabilistic constraint and a new objective function [89].

4. For linear programs with ν having a continuous distribution, build an equivalent problem using the basis which are primal feasible for the problem.

The method we consider here belongs to the third category of approaches, with the first model introduced by Kataoka in 1963 [89] which studied the case of a multivariate distribution for ν. Let us consider the program:

$$\begin{aligned} \min_{x} \quad & z = h\left(x, \nu\right) \\ \text{s.t.} \quad & x \in D \end{aligned} \tag{2.10}$$

with D the set of constraints. Solving 2.10 consists in finding an x^* such that $z^* = h(x^*, \nu) \leq z = h(x, \nu),\ \forall x \in D$. If we are interested in a robustness on the result with a guaranteed probability $1 - p$, the following stochastic model should be solved:

$$\begin{aligned} \min_{x} \quad & u \\ \text{s.t.} \quad & \mathbb{P}\left(h(x, \nu) \leq u\right) \geq 1 - p & (2.11) \\ & x \in D & (2.12) \end{aligned}$$

where $p \in (0, 1)$ is a probability threshold and D the set of initial constraints. This means finding x^* such that $u(x^*) \leq u(x),\ \forall x \in D$ which is equivalent to searching for the superior bound of an unilateral confidence interval for h such that $x \in D$. Of course, the above approach makes sense in situations when the central limit theorem cannot be applied. If the number of random variables ν is large enough and the magnitude of the variation is not too large, a more appropriate choice seems to make use of the expectation $E(h(x, \nu))$.

For a general problem with uncertainty both in objective function and in constraints, we consider that the robust binomial approach could be easily combined with the above stochastic model, if the random variables affecting the constraints are independent from those appearing in the objective function. While the probability in constraints is treated via the robust binomial approach which determines the feasible set D, for guaranteeing the result with a high probability, we are looking for an upper bound for the confidence interval of the objective function.

Conclusion

Taking into account uncertainty is a crucial aspect when solving an optimization problem, which has a great influence on the quality of the overall solution. As such, optimization under uncertainty and in particular stochastic programming and robust optimization remain between the most active domains of research nowadays. Nevertheless, due to the difficulties related to solve problems with uncertain parameters, most of the existing approaches make simplifying assumptions which are not always consistent with the real data.

The robust binomial approach takes advantage of experimental data and reinterprets the scenario approach (between the only existing methods without particular assumptions about the distribution of random variables) with tools from statistical hypothesis. As such, for a sample of size NS, it provides a safe approximation, guaranteed with a confidence level $1 - \alpha$ (with $\alpha \in (0, 1)$), to chance constrained programs with a required $1 - \varepsilon$ threshold on the probability constraints. We also present the necessary steps for integrating the robust binomial approach in an existing approximate algorithm (in the admissibility oracle). The founding principles of the robust binomial approach and a case study consisting in stochastic partitioning of process networks are subject of the journal paper [150].

Next chapters are dedicated to the application of the robust binomial approach to optimization problems occurring in the compilation process of a dataflow application for manycore. Nevertheless, due to its simplicity and the reduced number of assumptions it requires, the robust binomial approach along with the methodology of solving an uncertain optimization problem from Chapter 2 are generic and thus, applicable to other domain fields.

Graph partitioning under uncertainty

Contents

This chapter is dedicated to the problem of stochastic partitioning of process networks, arising in the resource allocation step of the compilation process of a dataflow application for manycore systems (see Chapter 1 for more details on the context).

Our interest is mainly in demonstrating the practical relevance of solving a stochastic problem by integrating the robust binomial approach into an existing heuristic developed for the deterministic case (as we conceptually explained in 2.3). As such, we want to show that having at our disposal an algorithm for the deterministic case, it is relatively easy in terms of software engineering (notably) to adapt it to the chance constrained version of the same problem. In the latter case, the solutions found are of consistent quality (with respect to the ones provided by the original algorithm) and more importantly, guaranteed to be robust to data variations with a confidence level of $1 - \alpha$ and a required probability level of $1 - \varepsilon$.

Since a multi-start constructive algorithm was already developed for partitioning networks of processes, we took advantage of the existing implementation in order to adapt the admissibility oracle and solve the stochastic case. The original greedy algorithm for the deterministic problem is given in section 3.3.2 and the associated computational results are presented in section 3.5. Therefore, we are not claiming that this existing algorithm is a best-in-class graph partitioning algorithm. What we do claim is that, using a slight adaptation of this algorithm, we can easily obtain robust solutions. Thus our experiments focus on showing that the algorithm for the stochastic version provides results consistent with those of the original one and attempt to quantify the "price of robustness".

3.1 Problem statement

We begin by a formal description of the problem of process networks partitioning:

As mentioned in Chapter 1, a dataflow application can be modeled as a directed graph in which the vertices are the tasks and the arcs are the channels. One of the (numerous) tasks of a dataflow compilation chain consists in mapping this graph onto the hardware resources of the target microprocessor architecture. Thus, in this chapter, we study the problem of assigning the weighted vertices of such a graph to a fixed set of partitions. We aim to minimize the sum of costs for edges having their extremities in different partitions (representing the processors), without exceeding the limited capacity (e.g. the memory footprint) of each partition. This application is an extension of the more abstract NP-hard problem of Node Capacitated Graph Partitioning (NCGP) [72], [64].

Let $G = (V, A)$ be a directed graph where the set of vertices $V = \left\{v_1, v_2, \ldots, v_{|V|}\right\}$ represents the tasks and the arcs $(v, w) \in A$ correspond to the channels of a process network. Let N be the set of disjoint nodes on a parallel architecture on which we want to map our graph. The resources (essentially memory footprint and computing core occupancy resources) are given by the set R and the capacities of the nodes are given by the multi-dimensional array $C \in \mathbb{R}^{+|R|}$. For the sake of simplicity, our study was limited to the case of homogeneous nodes: i.e. all nodes have the same capacity.

Let us also define two functions. $s : V \longrightarrow \mathbb{R}^{+|R|}$, is the size function for the vertex weights, with $s(v)_r$ being the weight of vertex v for resource r. The second function, defined for the edges, is the affinity function $q : A \longrightarrow \mathbb{R}^{+|R|}$ where $q((v, w)) > 0$ denotes the weight of edge $(v, w) \in A$ and $q((v, w)) = 0$ if no edge (v, w) exists between the vertices v and w. In the remaining, we will use the following simplified notation: $Q_{vw} = q((v, w))$ for each arc $(v, w) \in A$ and $S_{vr} = s(v)_r$, for $r \in R$ and $v \in V$.

The partitioning problem we work on consists in finding an assignment of vertices to nodes, denoted $f : V \longrightarrow N$, that satisfies the capacity constraints for all resources:

$$\sum_{v \in V : f(v) = n} S_{vr} \leq C_r, \forall n \in N, \forall r \in R, \tag{3.1}$$

by minimizing the objective function:

$$\sum_{(v,w) \in A : f(v) \neq f(w)} Q_{vw}$$

The stochastic case we consider for the graph partitioning with capacity constraints is taking into account the uncertainty affecting the node weights, being a relatively novel problem, as we will show in the following.

3.2 Related works

3.2.1 Deterministic graph partitioning

Since the graph partitioning problem and especially the bisection problem (a particular version of the problem for $|N| = 2$, also NP-hard) have been of great interest in the past, many different resolution methods were developed for treating the deterministic case. There are several surveys (see [66], [59], [18]) resuming the existing algorithms for deterministic graph partitioning.

Due to the NP-hardness of graph partitioning, the literature addressing the exact resolution of this problem is relatively sparse. Among the most successful exact deterministic approaches are the branch-and-price or column generation methods [86], [119]. Interesting results are also obtained in [64], in which the polyhedral structure of the problem is analyzed and classes of strong valid inequalities are included in a branch-and-cut algorithm. We should also mention the existence of a few approaches exploiting lower bounds for the problem. Particularly new lower bounds of rather good quality were found using semidefinite programming [103] as well as multi-commodity flows [140]. Nevertheless, these exact methods can handle only relatively small graphs and are too slow to be applied to larger graphs (with, for example, more than a thousand vertices). Mainly for this reason, these methods are not adequate to our application where we have to partition instances with a number of vertices varying roughly between 500 and 4000 on 16 to 64 nodes.

Therefore, we turn our attention to heuristics, the usual and more practical methods for tackling such problems. There are a large number of such methods, either global or local, that differ with respect to cost (time and memory space required to run the algorithm) and partition quality, i.e. the optimal solution or the cut size. One of the earliest and most popular algorithms, due to Kerninghan and Lin [90], originally proposed for the bisection case, is of quite high complexity (for a graph with $|E|$ edges, $O(|E|)$ for Fiduccia adaptation [65] or in the original version $O(|E|^2 log(|E|)))$. Also, it demands a lot of computational effort for being adapted to the capacitated generalized problem. Among local metaheuristics, one of the most used to solve the graph partitioning problem is simulated annealing, mainly because of its simplicity [85], [93]. However, it highly depends on the structure of the problem and for large sized instances, the required execution time may become prohibitive. For very large graphs, rather good results were found by global approaches, such as the multilevel and hierarchical methods [78], [88] or the more recent method of fusion-fission [17].

3.2.2 Stochastic graph partitioning

Previous work related to the stochastic form of the problem treated in the present dissertation is quite scarce. Fan et Pardalos studied a problem relatively close to ours: partition the vertex set of a graph into several disjoints subsets so that the sum of weights of the edges between the disjoint subsets is minimized, with a cardinality constraint on each subset and the uncertainty affecting the edge weights. In [61], assuming there is no information on the probability distribution other than that the

weights on the links are independent and bounded in known intervals, they formulate the problem using a robust optimization model, similar to [14]. The equivalent linear programming formulation is then solved by an algorithm based on a decomposition method. In a more recent study [62], they introduce the two-stage stochastic graph partitioning, assuming that the distribution of edge weights has finite explicit scenarios. Having as objective to minimize the expected weight of edges in the set of cuts over all scenarios, they present a nonlinear stochastic mixed integer model and propose an equivalent integer programming formulation for solving the problem using CPLEX. Taskin et al. [32] study the stochastic edge-partition problem, where the edge weights are uncertain, and are realized only after the node-to-subgraph assignments have been made. They introduce a two-stage cutting plane algorithm with integer variables in both stages and, to overcome the computational difficulties, they also prescribe a hybrid integer/constraint programming method as an alternative.

The approaches above differ in several aspects from our study. First, in our case, the problem formulation is not the same, dealing with multidimensional capacity constraints on the nodes instead of cardinality constraints. Consequently, uncertainty is addressed in a different manner, the assumption of uncertainty being made on the weights of the vertices rather than on the weights of the edges. Finally, we remark that the existing methods are exact and thus, mostly suited for small-size instances of the problem, the numerical experiments being performed on graphs with at most 100 vertices. On the contrary, for the processes placement problem, we are interested in practice to partition much larger graphs.

3.3 Preliminaries: Deterministic algorithm

3.3.1 Relative affinity

Before describing the randomized greedy heuristic our stochastic algorithm is based on, let us recall the notion of relative affinity, initially introduced in [47] (see also [149]).

Let S and T be two disjoint subsets of V.

Definition 3.1. *The affinity of S for T is given by :*

$$\alpha(S,T) = \sum_{(v,w)\in\delta(S,T)} Q_{vw}.$$

with $\delta(S,T) = \{(v,w) : v \in S; w \in T\}$. It follows that $\alpha(S,T) = \alpha(T,S)$.

Definition 3.2. *The total affinity of S (similarly for T) is given by*

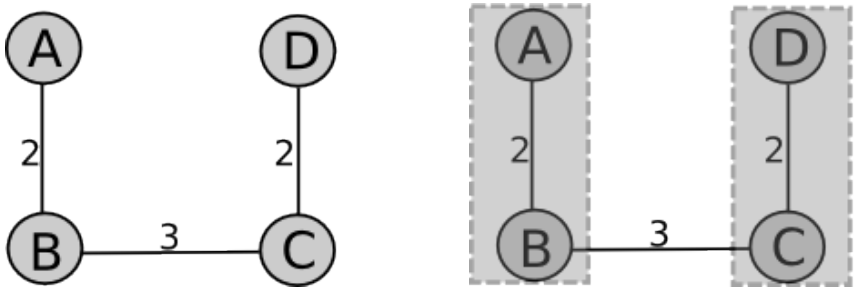
$$\beta(S) = \alpha(S, V \setminus S).$$

Definition 3.3. *The relative affinity of S for T is defined as*

$$\gamma(S,T) = \frac{1}{2}\alpha(S,T)\left(\frac{1}{\beta(S)} + \frac{1}{\beta(T)}\right)$$

where $\frac{\alpha(S,T)}{\beta(S)}$ *represents the contribution to the total affinity of* S *of the edges adjacent to* S *and* T.

Let us illustrate these notions through a simple example [47] on the undirected graph shown in Figure 3.1a. We suppose that we only have one resource and that all the vertices have unitary weights and we want to partition the graph into two nodes of capacity equal to 2. A greedy partitioning using the total affinity would have begun by putting together the vertices B and C, resulting in a solution of cost 4. Instead, a greedy partitioning based on relative affinity would match the vertices A and B (and C and D), with $\gamma(\{A\},\{B\}) = \gamma(\{C\},\{D\}) = 0.7$ and $\gamma(\{B\},\{C\}) = 0.6$, obtaining a solution of cost 3 (see 3.1b).



(a) (b)

Figure 3.1: (a) A graph example (b)2-partition using the relative affinity [47]

3.3.2 Randomized greedy algorithm: deterministic case

In order to fully illustrate our methodology for leveraging an existing algorithm solving the deterministic version of a problem to the stochastic case let us describe our starting point. (Note that we do not claim to be the ultimate graph partitioning heuristic, emphasis being made on the leverage-for-the-stochastic-case methodology.)

Initially described in [146], the randomized greedy algorithm we are starting from is based on the *relative affinities* of *admissible assignments* and *admissible fusions*.

Let W be a set of vertices not yet assigned to a node.

Definition 3.4. *An assignment of vertex* v *to node* n *is admissible if it satisfies the capacity constraints for node* n*, such that for every resource* $r \in R$*:*

$$S_{vr} + \sum_{w \in V \setminus W : f(w) = n} S_{wr} \leq C_r$$

Definition 3.5. *A fusion between the nodes* n *and* m *is admissible if for every resource* $r \in R$*:*

$$\sum_{v \in V \setminus W : f(v) = n} S_{vr} + \sum_{v \in V \setminus W : f(v) = m} S_{vr} \leq C_r$$

The assignments are favored over fusions and, when tie-breaking with respect to relative affinity, the heuristic prioritizes the assignment of vertices with heavier weights on less loaded nodes and the fusion of the most loaded nodes. We also formally define the relations of *heavier vertex* and *more loaded node* which are being used in the algorithm for the multidimensional case.

Definition 3.6. *The vertex v is smaller or lighter than the vertex w if:*

$$\max_{r\in R}\frac{S_{vr}}{C_r} < \max_{r\in R}\frac{S_{wr}}{C_r} \tag{3.2}$$

Definition 3.7. *The node n is more loaded than the node m if:*

$$\max_{r\in R}\frac{1}{C_r}\left(C_r - \sum_{v\in V\setminus W: f(v)=n} S_{vr}\right) < \max_{r\in R}\frac{1}{C_r}\left(C_r - \sum_{v\in V\setminus W: f(v)=m} S_{vr}\right)$$

The algorithm, to which we will refer as RG_PART, takes as input the set of unassigned vertices W (initially equal to V), the set of nodes N, the set of resources R and the vertex weights S_{vr}. A basic version of the algorithm is given underneath.

Algorithm 3.1: RG_PART

Input: W, N, R, S_{vr} for each $\{v \in V, r \in R\}$

1: Initialization $W = V$

2: Assign the first $\min(|V|, |N|)$ vertices in lexicographic order to the $|N|$ nodes and update the set W

3: Find an admissible assignment (v^*, n^*) ($v^* \in W$, $n^* \in N$) cf. Def. 3.4, if any, with maximal relative affinity:

$$\gamma_1 = \gamma(\{v^*\}, \{v \in V \setminus W : f(v) = n^*\})$$

4: Find an admissible fusion (n_1^*, n_2^*) ($n_1^* \in N$, $n_2^* \in N$) cf. Def. 3.5, if any, with maximal relative affinity:

$$\gamma_2 = \gamma(\{v \in V \setminus W : f(v) = n_1^*\}, \{v \in V \setminus W : f(v) = n_2^*\})$$

5: If $\gamma_1 \geq \gamma_2$ then assign v^* to n^* and update the set W. Else merge n_1^*, n_2^*

6: If $W = \emptyset$ or there is neither any admissible assignment nor any admissible fusion, stop. Else, go to Step 3.

Output: assignment f

Since greedy algorithms tend to sometimes get trapped with poor quality solutions, a type of diversification strategy is required. This is the reason why a randomized version of the algorithm is executed several times (i.e. in a *a multi-start* fashion). The randomization strategy consists in executing the algorithm first on the list of vertices sorted by their decreasing weights (see step 2 of the algorithm and for multi-resource case, Eq. 3.2) and several times afterwards using randomized versions of the list of vertices.

The algorithm being given for the deterministic version of our problem, we can now turn to the case we are interested in, the one in which the weights of the vertices are uncertain.

3.4 Chance constrained version

The robust binomial approach described in chapter 2 can be easily applied for solving the stochastic version of the capacitated graph partitioning problem. All we have to do is to combine the statistical hypothesis testing with the heuristic algorithm RG_PART by counting the number of times the constraints are violated by an initial sample.

For the stochastic version of our graph partitioning problem, formally stated in section 3.1, we make the assumptions that the task weights S_{vr} are random variables and that we dispose of a relevant sample of NS independent and identically distributed realizations of the uncertain vector of task weights. For $i = 1$ to NS, let $\tilde{S}_{vr}^{(i)}$ be the realization of the i-th observation for resource r.

Let us also note the event $e_{nr} = \{\sum_{v \in V : f(v)=n} S_{vr} \leq C_r\}$. The capacity constraint, expressed for the deterministic case in equation (3.1), becomes:

$$\mathbb{P}\left(\bigwedge_{n \in N} \bigwedge_{r \in R} e_{nr}\right) \geq 1 - \varepsilon.$$

In order to ensure that the probabilistic constraint is satisfied with a given confidence level at every step of the algorithm, it is necessary to redefine the notions of *admissible assignment* and *admissible fusion*.

Definition 3.8. *An assignment of vertex v to node n is stochastically admissible if the sum:*

$$\sum_{i=1}^{NS} \chi(\{\exists n' \neq n, \exists r : \sum_{w : f(w)=n'} \tilde{S}_{wr}^{(i)} > C_r\} \vee \{\exists r : \tilde{S}_{vr}^{(i)} + \sum_{w : f(w)=n'} \tilde{S}_{wr}^{(i)} > C_r\}),$$

is less than $NS - k(NS, 1-\varepsilon, \alpha)$, where $\chi(\mathcal{P}_a) = 1$ if and only if the predicate $\mathcal{P}_a$ is true.

This calculation can be simplified by using an ad hoc data structure, a boolean bi-dimensional array of size $|N| \times NS$, denoted t, indicating for the partial current partitioning if, for every node, the sample i has already induced a violation.

Thus, the assignment of a vertex v to a node n is *stochastically admissible* if:

$$\sum_{i=1}^{NS} \chi(t[n', i] \vee \{\exists r : \tilde{S}_{vr}^{(i)} + \sum_{w : f(w)=n'} \tilde{S}_{wr}^{(i)} > C_r\}) < NS - k(NS, 1-\varepsilon, \alpha)$$

With the use of the boolean array, the computation of an admissible assignment increases in complexity linearly, with a factor of NS, compared to the deterministic case.

If the vertex v is effectively assigned to node n then the boolean array is updated with:

$$t[n,i] := t[n,i] \vee (\exists r : \tilde{S}_{vr}^{(i)} + \sum_{w:f(w)=n} \tilde{S}_{wr}^{(i)} > C_r)$$

Definition 3.9. *A fusion between nodes n and m is stochastically admissible if the sum:*

$$\sum_{i=1}^{NS} \chi(\{\exists n', r : \sum_{w:f(w)=n'} \tilde{S}_{wr}^{(i)} > C_r\} \vee \{\exists r : \sum_{w:f(w)=n} \tilde{S}_{wr}^{(i)} + \sum_{v:f(v)=m} \tilde{S}_{vr}^{(i)} > C_r\}),$$

is less than $NS - k(NS, 1-\varepsilon, \alpha)$, where $\chi(\mathcal{P}_f) = 1$ if and only if the predicate $\mathcal{P}_f$ is true,.

Analogously, we can simplify $\mathcal{P}_f$ by using the same boolean matrix $|N| \times NS$. Once the fusion is realized, the entries for nodes n and m are updated as follows:

$$\begin{aligned} t[n,i] &:= t[n,i] \vee (\exists r : \sum_{w:f(w)=n} \tilde{S}_{wr}^{(i)} + \sum_{v:f(v)=m} \tilde{S}_{vr}^{(i)} > C_r) \\ t[m,i] &:= false \end{aligned}$$

As for the computation complexity, we remark a linear increase with a factor of NS in comparison to the deterministic version.

Also, since we have to deal with a sample of size NS, we can redefine the way we compare the vertices and the nodes weights, by taking into account the average over all realizations as follows.

Definition 3.10. *The vertex v is smaller or lighter in average than the vertex w if:*

$$\max_{r \in R} \frac{\sum_{i=1}^{NS} \tilde{S}_{vr}^{(i)}}{NS * C_r} < \max_{r \in R} \frac{\sum_{i=1}^{NS} \tilde{S}_{wr}^{(i)}}{NS * C_r}$$

Definition 3.11. *The node n is more loaded than the node m in average if:*

$$\max_{r \in R} \frac{1}{C_r} \left(C_r - \frac{\sum_{v \in V \setminus W : f(v)=n} \tilde{S}_{vr}^{(i)}}{NS} \right) < \max_{r \in R} \frac{1}{C_r} \left(C_r - \frac{\sum_{v \in V \setminus W : f(v)=m} \tilde{S}_{vr}^{(i)}}{NS} \right)$$

The above definitions can then be easily integrated in the algorithm described in section 3.3.2, without any major destructuring. As such, the algorithm Alg.3.2, named RG_PART_STOCH, is used for solving the chance constrained version of the Node Capacitated Graph Partitioning problem.

It should be noted that the only remarkable differences between the algorithm RG_PART and its stochastic counterpart RG_PART_STOCH are in Step 3 and Step 4 when deciding if the current assignment or fusion is admissible. Additionally, the algorithm for the chance constrained case needs as input the NS realizations of S_{vr}, the tasks weights for each resource, $1-\varepsilon$ the prescribed probability level and $1-\alpha$ the confidence level.

By using the robust binomial approach within a heuristic approach, we also overcome the computational effort of taking into account the uncertainties of the weights of the vertices. We could even further improve the performances of the heuristic by parallelizing the computations of admissible assignments and of admissible fusions.

Algorithm 3.2: RG_PART_STOCH

Input: W, N, R, ε, α, NS, $\tilde{S}_{vr}^{(i)}$ for each $\{v \in V,\, r \in R,\, i = 1...NS\}$

1: Initialization $W = V$

2: Assign the first $\min(|V|, |N|)$ vertices in lexicographic order to the $|N|$ nodes and update the set W

3: Find an admissible stochastic assignment (v^*, n^*) ($v^* \in W$, $n^* \in N$) cf. Def. 3.8, if any, with maximal relative affinity:

$$\gamma_1 = \gamma(\{v^*\}, \{v \in V \setminus W : f(v) = n^*\})$$

4: Find an admissible stochastic fusion (n_1^*, n_2^*) ($n_1^* \in N$, $n_2^* \in N$) cf. Def. 3.9, if any, with maximal relative affinity:

$$\gamma_2 = \gamma(\{v \in V \setminus W : f(v) = n_1^*\}, \{v \in V \setminus W : f(v) = n_2^*\})$$

5: If $\gamma_1 \geq \gamma_2$ then assign v^* to n^* and update the set W. Else merge n_1^*, n_2^*.

6: If $W = \emptyset$ or there is neither any admissible assignment nor any admissible fusion, stop. Else, go to Step 3.

Output: assignment f

3.5 Computational results

In this section, we report on the computation experiments of applying the above sample-based randomized greedy heuristic to the chance constrained version of graph partitioning with uncertainty affecting the weights of the vertices. All these experiments have been carried out on a Linux PC workstation, with a 3.80 GHz Pentium(R) processor, 3 GB of memory and Ubuntu 10.04 as operating system. In the rest of the section, we report about the benchmark, the random variables used in our computation and different evaluation measures. Then we discuss the results of the heuristic for the chance constrained version in comparison with the heuristic for the deterministic case.

3.5.1 Benchmark and Uncertain Parameters Generation

Since, to the best of our knowledge, there are no probabilistic instances defined for the graph partitioning problem with uncertain weights on the nodes, we tested our algorithm on two modified sets of test problems, originally intended for the deterministic case.

The first set of instances consists of some examples of grids, representative in size for our application. Besides, these instances are easy to modify and we can use them to test different configurations of the parameters for our method. The second set is defined by instances publicly available defined in Johnson et al. [86] and initially used for bisection. The tests on this second set were performed in order to confirm the effectiveness of our stochastic algorithm (both in terms of solution quality and running time) on a set of representative instances.

It should be noted that the instance "Grid 23x23", from the first data set, with 529 vertices and 16 nodes, is the closest in size to the real instances we have to deal with in our application context, at least as a first step.

The number of vertices for Johnson instances varies between 124 and 1000 and, for both sets, we consider the case of mono-dimensional resources.

In the deterministic case, the tests were performed for unitary weights for edges and vertices.

We have generated the random variables representing the weights of the vertices by simulating a joint bimodal distribution. The two modes are uniform in their intervals and selected in an equally likely manner.

The first mode is represented by the hypercube:

$$[0.8, 0.9]^{|V|},$$

and the second one, by the hypercube:

$$[1.1, 1.2]^{|V|}.$$

3.5.2 Results for the deterministic version

Table 3.1 shows the experimental results obtained by applying the RG_PART heuristic for graph partitioning on the first data set with deterministic vertices weights. All the results were obtained for the monodimensional case (the capacity of each node is indicated in column "C") with unitary weights for edges and vertices. The column "Multi" in Table 3.1 shows the solutions found by running the multi-start version of the heuristic (with 10 iterations) and the column "Time" shows the running time for one iteration in average over 10 iterations.

Table 3.1: Computational results of RG_PART heuristic for deterministic case: grid problems

Inst.	#Vertices	#Nodes	C	Multi	Time (sec.)
Grid 4×4	16	4	4	8	$\approx$0
Grid 10×10	100	5	20	28	$\approx$0
Grid 23×23	529	14	40	150	0.12

The RG_PART heuristic was applied on the larger sizes instances of Johnson et al. [86], with unitary weights for edges and vertices. As illustrated by Table 3.2, the solutions are reasonably close to the optimum (*) or to the best known solutions (column "Best known"). Furthermore, for most instances we observed that the solutions values found have an average differential approximation ratio [50] of 5.22% compared to the best known value.

Although these results are only of moderate quality, our goal in this experimental part is to provide them for serving, in the next section, as a starting point for measuring "the price of robustness" of the solutions obtained by the algorithm derived for the stochastic case.

Table 3.2: Computational results of RG_PART heuristic for deterministic case: Johnson instances

Name	$\|V\|$	C	Best known	Multi
Gsub.500	500	250	206	236
G1000.0025	1000	500	95	118
G1000.005	1000	500	445	509
G1000.01	1000	500	1362	1461
G1000.02	1000	500	3382	3526
G124.02	124	62	13*	15
G124.04	124	62	63*	68
G124.08	124	62	178	183
G124.16	124	62	449	471
G250.01	250	125	29*	36
G250.02	250	125	114	127
G250.04	250	125	357	378
G250.08	250	125	828	855
G500.005	500	250	49*	61
G500.01	500	250	218	253
G500.02	500	250	626	669
G500.04	500	250	1744	1825
U1000.05	1000	500	1*	6
U1000.10	1000	500	39*	69
U1000.20	1000	500	222	299
U1000.40	1000	500	737	866
U500.05	500	250	2*	12
U500.10	500	250	26*	68
U500.20	500	250	178*	196
U500.40	500	250	412	412

3.5.3 Results for the chance constrained version

We have tested our adaptation of the algorithm for the stochastic case on the same problems varying the parameters ε and α in the range $\{0.01, 0.05\}$. To obtain a set of stochastic instances, we have considered that the weights of the vertices are random variables with the aforementioned bimodal distribution and we generated corresponding samples of size 100 and respectively 1000. Choosing a smaller size for the sample may make the solution infeasible, and larger values of NS increase computation time of the problem.

The method has been implemented in C language and, for each instance, 10 random iterations of our algorithm were executed. Tables 3.3 - 3.5 summarize the numerical results for the grid problems for different values of the parameters NS, ε and α. The computational results for the second data set, the Johnson instances, are reported in Tables 3.6 - 3.8.

For each instance from the data sets, we performed two tests. The first test consists in keeping the same node capacity as for deterministic case (see columns C from Tables 3.1 - 3.2) and progressively increasing the number of nodes used in the deterministic case until the probabilistic constraint is satisfied.

The numerical results of this test, reported in section "1st test" of Tables 3.3 - 3.8 are: the minimal number of nodes for which the probabilistic constraint is respected (column "#nodes"), the solution value (column "sol") and the average execution time for 10 iterations (column "time").

For the second test, we maintain the same number of nodes as in the deterministic case, but we gradually increase the capacity of all nodes (starting from one used in the deterministic case) until finding a feasible solution, satisfying the probabilistic constraint.

The results of this second test, reported in section "2nd test" of Tables 3.3 - 3.8 are: the minimal capacity of each node for which we obtain a feasible solution (column "C"), the solution value (column "sol") and the average execution time for 10 iterations (column "time").

Table 3.3: Computational results of the stochastic method for $NS = 100$, $\varepsilon = 0.05$, $\alpha = 0.05$: grid problems

	1st test			2nd test		
Name	#nodes	sol	time	C	sol	time
Grid 4×4	6	14	≈ 0	4.71	12	≈ 0
Grid 10×10	6	38	0.02 s	23.3	29	0.01 s
Grid 23×23	16	182	1.12 s	44.1	173	0.99 s

It is worthwhile noting that the solutions obtained in the second test, by increasing the node capacity, are of better quality than the solutions of the first experiment (see columns "sol") and can be adjustable more accurately. For example, in Table 3.4 for Grid 10×10, for finding a feasible solution, we must add two more nodes. However, in this case the solution found is too conservative since all the constraints are verified. Since, however, in practice it is easier to modify the number of nodes

Table 3.4: Computational results of the stochastic method for $NS = 1000$, $\varepsilon = 0.05$, $\alpha = 0.05$: grid problems

	1st test			2nd test		
Name	#nodes	sol	time	C	sol	time
Grid 4×4	6	14	≈ 0	4.712	12	≈ 0
Grid 10×10	6	37	0.16 s	23.273	37	0.13 s
Grid 23×23	16	182	11.23 s	44.13	172	9.65 s

Table 3.5: Computational results of the stochastic method for $NS = 1000$, $\varepsilon = 0.01$, $\alpha = 0.01$: grid problems

	1st test			2nd test		
Name	#nodes	sol	time	C	sol	time
Grid 4×4	6	14	≈ 0	4.74	10	≈ 0
Grid 10×10	6	37	0.15 s	23.36	37	0.13 s
Grid 23×23	16	182	10.75 s	44.183	193	9.67 s

than the capacity of each node, we also investigated the results found by the first test.

Our main purpose with these tests is to get an idea of the cost of the robustness of the solutions, independently of concrete application constraints.

In evaluating the performance of our heuristic method, between the main aspects we consider are: the capacity and the number of nodes needed for finding a feasible solution, the time factor and the robustness and quality of the solutions.

In our first test, we were interested in the number of nodes needed for the stochastic case compared to the deterministic one. Our computational results show that the ratio between the number of nodes for stochastic partitioning and the number of nodes for deterministic partitioning for the same instance is 1.5, except for Grid 23×23, for which the ratio is equal to $\approx$1.14. The same ratio of 1.5 was found for the Johnson instances.

For the second test, we analyzed the required increase in capacity for solving the stochastic version of the problems. The stochastic solutions of the instances reported in Tables 3.3 - 3.5 are obtained for an equally large increase in the capacity of the nodes in the order of 1.1. For the Johnson instances, the capacity of nodes for stochastic partitioning is superior to the nominal capacity with $\approx$ 1.15. As one may expect, keeping the same probability and confidence levels and changing the sample size does not significantly affect the minimal capacity of the nodes for which a valid solution is found. On the contrary, imposing a higher probability and confidence levels demands a minimal capacity of nodes slightly larger (in the order of 0.001). Following the run of each instance, we have also observed a particular behavior consisting in a threshold effect of the solutions, sensible to the node capacity variations. One example is the problem U1000.10 for which an augmentation of the capacity from 576.06 to 576.20 results in a largely better solution (69 against 115).

Table 3.6: Computational results of the stochastic method for $NS = 100$, $\varepsilon = 0.05$, $1 - \alpha = 0.95$: Johnson problems

	1st test			2nd test		
Name	#nodes	sol	time	C	sol	time
Gsub.500	3	301	5,57	288,300	244	5,55
G1000.0025	3	135	58,97	575,800	131	70,11
G1000.005	3	649	62,23	575,900	513	72,10
G1000.01	3	1865	65,30	575,900	1456	77,36
G1000.02	3	4481	68,78	575,940	3579	74,10
G124.02	3	18	0,16	71,650	21	0,11
G124.04	3	91	0,16	71,650	72	0,11
G124.08	3	233	0,16	71,670	199	0,11
G124.16	3	585	0,16	71,680	475	0,12
G250.01	3	40	0,75	144,200	38	0,64
G250.02	3	162	0,76	144,260	128	0,63
G250.04	3	485	0,78	144,250	393	0,64
G250.08	3	1074	0,77	144,200	862	0,65
G500.005	3	68	5,14	288,370	67	5,20
G500.01	3	308	5,36	288,340	269	5,04
G500.02	3	860	5,42	288,280	679	5,44
G500.04	3	2287	5,56	288,270	1835	5,60
U1000.05	3	17	67,76	576,100	16	73,50
U1000.10	3	101	65,55	576,100	110	77,74
U1000.20	3	417	67,80	576,200	303	75,23
U1000.40	3	1370	68,26	576,300	1018	77,00
U500.05	3	10	5,05	288,390	7	5,27
U500.10	3	88	5,58	288,270	66	5,38
U500.20	3	278	5,49	288,200	396	5,43
U500.40	3	663	5,23	288,380	574	5,28

Table 3.7: Computational results of the stochastic method for $NS = 1000$, $\varepsilon = 0.05$, $1 - \alpha = 0.95$: Johnson problems

	1st test			2nd test		
Name	#nodes	sol	time	C	sol	time
Gsub.500	3	298	26,72	288,240	252	18,94
G1000.0025	3	136	139,30	576,030	134	119,48
G1000.005	3	653	143,70	576,060	528	123,52
G1000.01	3	1866	141,00	576,060	1470	125,70
G1000.02	3	4482	140,86	576,040	3599	127,95
G124.02	3	17	1,39	71,662	17	0,91
G124.04	3	87	1,37	71,654	68	0,92
G124.08	3	237	1,36	71,660	182	0,93
G124.16	3	599	1,35	71,653	479	0,91
G250.01	3	39	5,84	144,310	39	4,00
G250.02	3	163	5,81	144,310	129	4,04
G250.04	3	483	5,78	144,295	387	3,99
G250.08	3	1080	5,75	144,257	872	3,98
G500.005	3	69	26,67	288,240	68	18,88
G500.01	3	320	26,74	288,240	258	19,00
G500.02	3	853	26,87	288,250	668	19,15
G500.04	3	2283	26,76	288,250	1829	19,18
U1000.05	3	18	140,90	576,050	6	125,70
U1000.10	3	74	139,40	576,060	115	126,80
U1000.20	3	417	141,40	576,030	339	126,80
U1000.40	3	1370	143,51	576,080	1032	132,60
U500.05	3	16	26,73	288,300	2	19,49
U500.10	3	105	26,90	288,260	75	19,32
U500.20	3	289	27,15	288,250	289	19,21
U500.40	3	663	26,73	288,240	569	18,89

Table 3.8: Computational results of the stochastic method for $NS = 1000$, $\varepsilon = 0.01$, $1 - \alpha = 0.99$: Johnson problems

	1st test			2nd test		
Name	#nodes	sol	time	C	sol	time
Gsub.500	3	298	25,32	288,610	240	18,94
G1000.0025	3	137	141	576,470	132	121,51
G1000.005	3	654	140,77	576,520	519	127,54
G1000.01	3	1870	141,66	576,520	1467	125,74
G1000.02	3	4475	141,23	576,530	3544	128,78
G124.02	3	17	1,35	71,865	17	0,9
G124.04	3	87	1,34	71,825	73	0,92
G124.08	3	237	1,33	71,851	187	0,92
G124.16	3	599	1,33	71,831	484	0,91
G250.01	3	39	5,73	144,548	40	4
G250.02	3	163	5,73	144,530	132	4
G250.04	3	483	5,65	144,515	383	4,06
G250.08	3	1085	5,65	144,523	856	3,95
G500.005	3	69	25,33	288,490	68	19,38
G500.01	3	320	25,32	288,540	258	19
G500.02	3	853	25,2	288,530	687	19,6
G500.04	3	2283	25,25	288,520	1852	19,3
U1000.05	3	20	141,79	576,550	1	125,76
U1000.10	3	74	140,69	576,520	90	128,03
U1000.20	3	421	143,14	576,570	339	131,14
U1000.40	3	1376	145,14	576,580	1137	127,47
U500.05	3	16	26,73	288,570	2	19,17
U500.10	3	105	25,75	288,560	62	19,03
U500.20	3	289	25,4	288,560	289	19,15
U500.40	3	663	25,08	288,570	569	19,41

Concerning the time factor, the overall execution time of our method mainly depends on the number of vertices and on the size of the sample. We note that the running time needed to solve Johnson instances is considerably higher than the time required for the grid problems, the reason being the presence of instances of larger size (e.g., G1000.0025-G1000.02, U1000.05-U1000.40). As expected, the larger is the sample size, the higher is the computation time, with an average of 48.04 sec. for a sample size of 1000 (Table 3.7) against 25.93 sec. for a sample size of 100 (Table 3.6) for the second test. It should also be noted that the computation time for the first test is, in average, superior to the time for finding solutions in the second one. By comparison of Table 3.7 and 3.8, it appears that when a higher probability level ε and confidence level α are imposed, a slightly higher execution time is needed.

Although these results could be improved (e.g. by code optimization and parallelism), such execution durations are already acceptable in our application context with respect to the usual compilation duration of a dataflow process network on a many core architecture.

The running times found for the stochastic version of the algorithm confirm the theoretical remarks (see Section 3.4) on a linear increase in complexity with a factor of NS in comparison of the deterministic case.

In order to measure the quality and the robustness of the stochastic solutions, the algorithm RG_PART was re-run with the same input parameters as the ones found with the chance constrained method. We kept the same number of nodes and respectively the same capacity of each node as the ones for which the chance constrained methods found feasible solutions and we considered unitary weights for arcs and unitary weights for tasks (which is the expected value of the distribution of our uncertain data).

As expected, for the first test consisting in increasing the number of nodes, the quality of the stochastic solutions is almost always worse than for the deterministic version and than for the solutions found by the second test. One exception is the instance U500.05, from Table 3.6 but this result is assumed to be due to the heuristic nature of our approach, which, by construction, provides no guarantees with respect to monotony.

Instead, the stochastic solutions of the second test are quite often close in quality to the solutions found when running RG_PART algorithm. By analyzing Tables 3.6-3.7, for ε, $\alpha = 0.05$ we found out that there are 14 and respectively 15 instances with a gap in the stochastic solution quality of less than 5% from the deterministic solutions. When analyzing the results for a probability level of 0.99 and a level of confidence of 0.99 (Table 3.8), we remark a number of 14 stochastic solutions close (a relative 5% gap) to the deterministic ones.

By comparing the quality of solutions for different values of the input parameters (NS, ε, α) it comes out that for the same probability and confidence levels, the obtained solutions when varying the sample size are quite similar, revealing that the performance of our algorithm does not deteriorate as the number of samples increases. It should be noted that it is however necessary to determine the minimal size of the sample needed to solve the problem with the required probability level. The required sample size for ε, $\alpha = 0.01$, is at least 459, which justifies our choice not to conduct tests for these values on the samples of size 100.

Concerning the robustness of the solutions found by the presented approach, we measured the number of times the deterministic solution is not satisfied on the used samples. The percentage of samples on which the deterministic solution is not satisfying the capacity constraints 3.1 is, in average, for Tables 3.6-3.8 between 48,24% and 50.04%.

Analyzing the overall results, we observe that our stochastic heuristic confirms the capacity of computing good solutions, within an admissible average running time, even for large instances. The quality of the solutions is comparable to the deterministic case (i.e. the "price of robustness" is reasonable). Moreover we guarantee that our solutions are robust to the uncertainties affecting the weights of the vertices.

Conclusion

In this chapter, we addressed the stochastic problem of partitioning communicating networks of process, with a theoretical equivalent in the Node Capacitated Graph Partitioning problem. The objective is the assignment of processes corresponding to the dataflow of an embedded application to a fixed number of nodes (clusters) and minimizing the communications inter-clusters while respecting the capacity of each node. The uncertain random variables are the weights of the processes for which we dispose of a sample of size NS and thus, we want to make sure the capacity constraints are getting respected with a probability of at least $1 - \varepsilon$.

In order to solve this problem, the RBA approach introduced in Chapter 2 was integrated within a greedy algorithm, originally intended for the deterministic case. The criteria for defining an admissible assignment of a vertex (process) to a node and an admissible fusion of two nodes were modified to assure that the number of constraints respected by the sample are superior to the threshold k established in function of NS, ε and α (see section 2.3 from Chapter 2 for details).

Since the chance-constrained graph partitioning is a relatively new problem and no stochastic instances were available in the literature, we had to generate two benchmarks adapted from the instances for the deterministic version, by generating random variables following a bimodal uniform distribution.

Numerical results showed that the obtained solutions have often a quality consistent with those computed for the deterministic version. More importantly, the solutions found are robust and guaranteed with a preset statistical significance level, to hold to data variations affecting the constraints. We also showed that not taking into account the stochastic nature of our data and considering only the deterministic case may lead to non feasible solutions with quite high probability (in average 50% of cases). Furthermore, this approach can solve with an acceptable computation time problems close in dimensions to the real instances a compiler would have to treat.

Joint placement and routing under uncertainty

Contents

This chapter addresses the problem of application mapping, classified as "one of the most urgent problems to be solved for implementing embedded systems" [117],[115]. Several workshops dedicated to mapping applications onto multi-core systems have been created in order to move beyond state-of-art in this domain (e.g. M-SCOPES [43]). There are different mapping methodologies varying in function of application and architecture models, constraints and assumptions imposed by the system, the metrics to be optimized, information available about the platform, etc.

The purpose of this work is to propose a placement method for dataflow process networks (DPNs), which also takes into account the computation of routing paths. As such, it provides an alternative approach to the sequential placement and routing steps of the resource allocation compilation step of a ΣC program (presented in section 1.4.3.3), which targets specific application domains (e.g. multimedia and networking) characterized by high bandwidth demands. Even if the two sub-problems of tasks mapping and routing have already been addressed in the literature, the novelty of our method consists in treating together task mapping and routing, and thus, taking into account the routing when placing the networks of processes, without any particular assumption on the network (here a Network-On-Chip) topology, both for deterministic and stochastic cases.

Before presenting the GRASP algorithm we conceived for the deterministic case (section 4.3.1), section 4.2 presents some existing mapping approaches. By adapting the GRASP using the general methodology introduced in 2.3, we solve the stochastic version of the same problem in section 4.4. Computational results, using both synthetic and real benchmarks, are shown and analyzed in the last part. We begin by a formal description of the problem.

4.1 Problem statement

The optimization problem we study consists in placing *at design time* the tasks of a DPN onto the network of clusters, such that the total bandwidth is minimal and for each pair of communicating tasks, there is a shortest routing path between tasks situated on different clusters.

The clusterized architecture is represented by a directed graph $G = (N, A, R, B_a)$ with N the set of nodes (clusters) and A the set of arcs between nodes, corresponding to the NoC links. $B_a : A \longrightarrow \mathbb{R}$ describes the bandwidths between different clusters of the target architecture, with $B_a((n_i n_j)) > 0$ the maximal capacity for arc (n_i, n_j) and $B_a((n_i n_j)) = 0$ if nodes n_i and n_j are not connected. R is the set of resources (essentially memory footprint and computing core occupancy) we have at our disposal. The capacities of the nodes are given by a multi-dimensional array $C_n \in \mathbb{R}^{+|R|}$.

For the sake of simplicity, we restrain our study to the case of homogeneous nodes and arcs for G, hence we suppose all nodes have the same capacity C_{nr} for each resource $r \in R$ and all arcs have the same maximal bandwidth B_a.

Let $DPN = (V, E, S, Q)$ represent the network of processes with V the set of vertices (tasks) and E the set of communication channels. $S : V \longrightarrow \mathbb{R}^{+|R|}$, is a size function for the tasks, with s_{tr} being the weight of task t for resource r. The function $Q : E \longrightarrow \mathbb{R}$ characterizes the communication between tasks where $q_{t_i t_j} > 0$ denotes the weight of arc $(t_i, t_j) \in E$ and $q_{t_i t_j} = 0$ if no arc (t_i, t_j) exists between t_i and t_j.

Let $g : V \to N$ be a mapping of tasks to the nodes. As such, we are interested in finding an admissible routable assignment g of tasks to nodes that minimizes the cost of the inter-clusters communication:

$$\sum_{(tt') \in E : g(t) \neq g(t')} q_{tt'} \tag{4.1}$$

In the context of our present work, an *admissible assignment* is a mapping of tasks to nodes which satisfies the capacity constraints:

$$\sum_{t \in V : g(t) = n} s_{tr} \leq C_{nr}, \forall n \in N, \forall r \in R, \tag{4.2}$$

and furthermore, it ensures that there exists a feasible routing between every two communicating tasks:

$$\{\forall (t, t') \in E \text{ and } g(t) \neq g(t') \text{ and } q_{tt'} \geq 0\} : \exists \text{route(t,t')} \tag{4.3}$$

which route respects the maximal capacity B_a of the links of the network. As such, the last condition verifies if all the bandwidths can be accommodated across the network G without exceeding the maximal capacity of the arcs in terms of bandwidth.

In order to simplify communication protocols, the search of possible routes will be limited to a single unsplittable commodity flow using a shortest-path routing strategy.

In our approach, we prioritize the minimization of the bandwidths and we analyze the difference, for an obtained placement between the routing our algorithm is using and an ideal one (using a shortest-path strategy).

Since the tasks mapping is equivalent to the Quadratic Assignment Problem which is NP-hard [72] and the unsplittable flow problem can be restricted to the Directed Edge Disjoint Paths problem, also NP-hard [95], the joint problem is straightforwardly NP-hard in the strong sense.

Regarding the size of instances specific to the application context, our method has to be able to map networks of processes with a few hundreds tasks on architectures having at least a dozen of nodes. For an experimental validation of our approach, one of the benchmarks consists of data coming from a motion target dataflow expressed in ΣC (as described in section 1.4.2) which has to be placed on a NoC in the form of a bi-dimensional torus 4×4.

4.2 Related works

As shown in Fig. 4.1, there are different criteria for classifying mapping technologies in function of the target architecture, when the placement takes place (at run time or design time) or the hierarchy involved. For static mappings, the

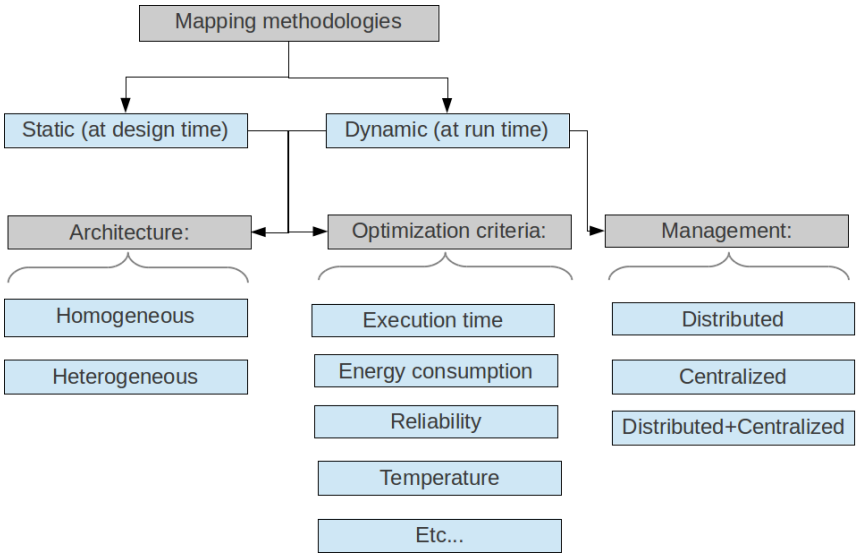


Figure 4.1: Classifying mapping approaches...

optimization is performed at design time while for dynamic workload scenario, the mapping takes place at run time. Moreover, for dynamic mappings, it is required a platform management responsible of mapping the tasks, scheduling, resource control, configuration control and task migration. Both design time and run time mappings can target either homogeneous or heterogeneous multi-cores systems and can be optimized for different optimization metrics.

The following section presents some of the methods belonging to these two categories for the deterministic case, with a particular emphasis on the static placement,

since this is the case we are most interested in. For a detailed survey on mapping strategies, please refer to [142].

4.2.1 Deterministic mapping

4.2.1.1 Static mapping

Static mapping methodologies are adapted to static workload scenarios for predefined applications with known behavior (in terms of computation and communication) and for fixed architectures. Since they are performed at design time and have a global view of the system, they can found better quality mappings compared with run time mapping (which explore only the neighborhoods of the mapped tasks).

Actually, even if most of the existing studies treating task mapping belong to this category, they remain different in the architectures they are targeting (homogeneous or heterogeneous), the optimization goal they fix and the restrictions they impose on the system. Moreover, we have to remark that even if the task mapping was and remains a relatively well studied problem (with the first works by Stone [152] and Lo [105]), the routing aspect has been often neglected, the scheduling problem drawing more the researchers attention. Also, usually, the objective of existing techniques is a placement with load balancing thus for which the tasks are equally distributed between all the processing elements (e.g.[157], [60], [55]) while for our current approach, the interest is in minimizing the number of used clusters.

For the mapping of applications expressed as dataflow process networks and for which the target architecture is a multi/manycore system, we can cite [127], [36], [22], [40], [71]. In [127], a simulated annealing algorithm is proposed for distributing Kahn Process Networks on Multiprocessors SoCs (MpSoCS) with at most four Processing Elements (PE) connected with dual shared bus. [71] proposes a parallelized simulated annealing approach for the DPNs mapping on a square torus architecture. Since this method is quite computationally demanding (roughly twenty minutes for a 31×31 square grid of tasks using 6 computing cores), it is more appropriate to be applied at the end of the development cycle of embedded applications. [36] presents an algorithm which, executed repeatedly, allows process and communication mapping of applications expressed as Kahn Process Networks onto a homogeneous MpSoC, with the objective of minimizing the application makespan. In [22], the authors address both mapping and scheduling of SDF applications on homogeneous multi-core platforms, using a Constraint Programming-based algorithm to maximize the throughput. Another method for solving the mapping and scheduling of a SDF application on a multi-core architecture, based on a genetic algorithm ([40]), takes into account the limited size of scratchpad memory (SPM) of the cores and tries to minimize the execution latency.

It is worth mentioning that the problem we address is different in constraints and objectives from the similar optimization problems occurring in VLSI (Very-Large-Scale-Integration) design flow for creating integrated circuits. In the latest case, the placement consists in taking a list of electronic components (which compose the circuit) and arranging them geometrically in a limited space while the routing is in charge of the design of the wiring connecting the placed components. The

result of the placement and routing (usually two steps realized sequentially) is called layout, which is the geometric description of the circuits parts and of the paths followed by the wires. Nowadays, the placement and routing for integrated circuits is usually made automatically with the help of EDA (Electronic Design Automation) tools (the most popular being those from Mentor Graphics, Cadence and Synopsys) and there are numerous dedicated algorithms (like FastPlace [155], FastRoute [130], ROOSTER [138] or IPR [131], just to name a few).

Several approaches [114], [147], [123], [81] for multi/manycore platforms propose configuration of the NoC according to the application in order to meet tasks requirements while fitting a specific SoC architecture. A branch-and-bound algorithm is proposed in [81] for the mapping of intellectual property blocks - IP (like CPU or DSP cores, video stream processors, input/output devices) on an architecture organized as regular tiles (composed of a processing core and a router), related by a NoC. The objective is to minimize the total energy spent on communication, by ensuring that *each IP goes to exactly one tile, no tile can host more than one IP* and having a routing constraint related to bandwidth usage. At each step, the algorithm assures a minimal and deadlock-free routing which respects the maximal load for each link of the NoC by incorporating a list of routing paths as part of the solution, instead of *a single routing path*. [123] conceives dynamic re-configuration mechanisms to match the NoC configuration to the communication characteristics of each use-case. A design methodology, restricted to Æthereal NoCs, is introduced for mapping, path selection and resource reservation in the network, by taking as input use-cases of the SoC. The objective of the mapping process is to design the smallest size NoC, with the smallest number of switches that satisfies the constraints for all use-cases. Instead, we consider that the manycore specification and in particular NoC characteristics such maximal bandwidth for links are rigid. As such, the placement and routing of tasks are realized afterwards (without worrying about scheduling) during the compilation process of a dataflow application.

Between the only approaches similar treating the same problem under same constraints as ours of which we are aware of is [144] which solves the problem as a master(placement)/slave(routing) couple. As such, the overall problem is split into two sub-problems, less complex. The assignment is solved using a semi-greedy algorithm while the routing paths are computed optimally with a mixed linear integer programming. However, the sequential resolution can un-structure the initial problem and the found placement may not be routable so there can be feasibility issues for the routing problem downstream as a result of relaxing some constraints for the upstream problem. The typical example consists of a placement non routable we cannot route because the flows between the nodes of the network exceed the maximal bandwidth capacity for the links B_a.

4.2.1.2 Dynamic mapping

In contrast with static approaches, dynamic mapping is performed at run time and as a consequence, the time taken by the mapping algorithm to find a solution adds to the overall application execution time. As such, a compromise must be made between the quality of the solution and the running time. Greedy algorithms are

typically used to provide an efficient mapping, optimizing different criteria such as reliability, energy consumption, execution time, etc. Once tasks mapping is performed, task migration (consisting in relocation of tasks) is a popular technique to respond to possible changes occurring at run time (e.g. performance bottleneck, new application entering the system, etc.).

Besides being suited for dynamic workload scenarios (the number of tasks executing in parallel varies in time), dynamic mapping has several advantages such as: adaptability to changes in the amount of available resources, possibility to upgrade the system (with new applications or standards not known at design time) or capability to avoid defective processing cores.

The platform manager, responsible for handling the mapping, can use a centralized management approach (one single core for the whole platform), distributed management (several communicating cores for managing several regions-clusters of the platform) or a mixture of centralized and distributed management. The centralized manager is more adapted to small platforms since this type of management is not scalable and can become a hot spot while the distributed management can be employed for larger architectures.

Also, the mapping process can be performed entirely at run time (on-the-fly mapping) or by using design time analysis (DSE) results. On-the-fly mapping requires efficient heuristics, independent of the architecture and that can be used to assign tasks coming from new applications (unknown at design time). The mapping based on previously analysis results is possible for an application known at design time and selected by making use of light heuristics between a series of already computed assignments at design time and stored on the system. Therefore, the intensive computation analysis takes place at design-time, taking as input the application and the architecture descriptions and producing a number of possible mappings. Such type of task assignment, also called hybrid mapping, performs better than on-the-fly mapping but it is less flexible since it must be aware at design time of the application requirements.

For more details on existing dynamic mapping methodologies, we invite the interested reader to refer to [142].

4.2.2 Stochastic mapping

While there are quite numerous studies analyzing the stochastic behavior of task execution times for soft real-time applications (e.g. for scheduling purpose), there are almost no works on optimizing the design of an application and taking into account the fact that task execution times are stochastic.

In [113], stochastic mapping and priority assignment of graph tasks on a multiprocessor hardware architecture is performed via a tabu search heuristic with the goal to optimize the ratio of deadlines missed. The underneath assumption is that for each task and each processor, a set of execution time probability density functions is available.

Lombardi et al. [106] address the stochastic problem of allocation and scheduling of conditional tasks graphs (CTG) for multiprocessor platforms, by guaranteeing that for each run time scenario encapsulated by the graph, the temporal and resource

constraints are satisfied. As such, they are searching for an unique assignment of starting time and resources to tasks, minimizing the expected value of the communication cost. By analyzing the task graph, they propose an exact analytical stochastic formulation of the objective and solve the allocation using Integer Linear Programming and the scheduling with Constraint Programming.

[141] studies the static robust resource allocation to application for distributed systems that are periodic sensor-driven when the execution times of the applications are independent random variables. While the objective function consists of minimizing the period between sequential data sets produced by the sensors, the probabilistic constraint is on the performance characteristic of the system. In order to compute this probability and to make sure it is superior to a minimal QoS (Quality of Service), bootstrap or FFT (Fast Fourier Transform) methods are used and the obtained approximation of the cumulative density function is further employed by the four greedy heuristics the authors design.

We can then affirm that, to the best of our knowledge, the *stochastic problem of joint placement and routing of dataflow applications for manycore* has not been yet addressed in the literature. Let us now get back for a moment to the GRASP algorithm we conceived for the deterministic problem, which, due to the robust binomial approach, can be adapted to solve the stochastic case.

4.3 Deterministic algorithm

We recall that our work is concerning the static placement and routing of applications for embedded manycore in the context of an iterative compilation. The objective is to place the tasks of an application to the nodes of the network and in the same time, mono-route the flows on the Network-On-Chip. As such, in order to design a resolution method for the joint mapping and routing, an important aspect to decide is for which step of the development cycle of embedded applications this algorithm is intended. The beginning of the development of an embedded application requires a short programmer/target feedback loop when the programmer is able to obtain a first working version of the application with a well coarse-grained structure. Thus, the beginning of the cycle requires for fast heuristics and can accept solutions of moderate quality. At the end of the development cycle, since more human and computing times are invested (e.g. acceptable compilation times of up to one night), more fine-grained optimizations are afforded. Hence, at this point of the cycle, one can accept more computationally intensive algorithms and more powerful computer systems.

Other algorithmic aspects to be considered are the problem complexity and the size of real instances to deal with, both factors making the building of a tractable exact resolution for both mapping and routing difficult and inefficient.

As such, we turned our attention to approximate algorithms and in particular to the GRASP metaheuristic, which seems a more suited choice to tackle this problem especially for the beginning of the development cycle of an application.

4.3.1 GRASP & Preliminaries

Introduced in the nineties by Feo and Resende [63], GRASP (Greedy Randomized Adaptative Search Procedure) is a multi-start metaheuristic, each iteration involving two phases: construction and local search. The construction phase builds a feasible solution using a greedy randomized algorithm. During the local phase, the neighborhood of the current solution is investigated in the search of better solutions. At the end, the best overall solution is kept as the result.

Alg. 4.1 illustrates the main blocks of our GRASP method for finding routable mappings of tasks to clusters. The input parameters are the set of tasks V, the set of nodes N, the set of resources R, the maximum number of iterations to be performed and also the parameter k used for controlling the amount of randomness (this is the probabilistic aspect of the construction phase). The mapping g_c found by the construction phase is further exploited in local search phase and optimized. If the resulting mapping g of this post-optimization is better than the previous best mapping g_b then we update g_b.

Algorithm 4.1: GRASP for joint placement and routing

Input: V, N, R, k, MaxIterations
1: $g_b \leftarrow$ null
2: **for** $i = 1$ to MaxIterations **do**
3: $g_c \leftarrow$ construction_phase(V, N, R, k)
4: $g \leftarrow$ local_search_phase(g_c)
5: update best assignment g_b with g if needed
6: **end for**
Output: best assignment g_b

Before explaining in more details each one of the two stages of our approach, let us recall the notions of total and relative affinity, initially introduced in [47].

Let S and T be two disjoint subsets of V.

Definition 4.1. *The affinity of S for T is given by :*

$$\alpha(S,T) = \sum_{(v,w)\in\delta(S,T)} q_{vw}.$$

with $\delta(S,T) = \{(v,w) : v \in S; w \in T\}$. It follows that $\alpha(S,T) = \alpha(T,S)$.

Definition 4.2. *The total affinity of S (similarly for T) is given by*

$$\beta(S) = \alpha(S, V \setminus S).$$

Definition 4.3. *The relative affinity of S for T is defined as*

$$\gamma(S,T) = \frac{1}{2}\alpha(S,T)\left(\frac{1}{\beta(S)} + \frac{1}{\beta(T)}\right)$$

where $\frac{\alpha(S,T)}{\beta(S)}$ represents the contribution to the total affinity of S of the edges adjacent to S and T.

4.3.2 Construction phase

The greedy constructive method from the first step of our GRASP is inspired from an existing algorithm, initially used for partitioning networks of processes and which was based on the notion of relative affinity ([144], [149]). We modified it in order to deal with routing and we changed the randomization strategy to intensify the diversity of the solutions.

The main idea of our constructive algorithm is to verify at each step of the mapping, that the flows between the assigned tasks can be routed by making use of the previous computed flows and trying to find feasible paths for the new or modified flows. At each step of the mapping, the computation of new routing paths is realized through a single source shortest-path algorithm on a reduced graph G' obtained from the original network G and whose arcs are weighted with a residual capacity C_{r_a}.

Let $G' = (N, A')$ be the reduced graph with the same number of vertices N as G and A' the set of arcs in G weighted with a positive residual capacity.

Let F be the set of flows between tasks and for each flow $f \in F$, $s(f)$, $d(f)$ and $w(f)$ are respectively the source, the sink (or the destination) and the demand (the weight) for flow f.

Let $sp(f)$ be the shortest path in G' by which the flow f is accommodated. So $sp(f)$ is composed of a set of nodes $\{n_1, n_2, \ldots, n_m\} \in |N| \times |N| \times \ldots |N|$ with $m \in \{0, |N| - 1\}$, $n_1 = g(s(f))$ and $n_m = g(d(f))$, such that $\forall i = \{1, \ldots, m-1\}$, $\exists (n_i, n_{i+1}) \in A'$, $C_{r_{(n_i,n_{i+1})}} \geq w(f)$ and the length of this path is minimal.

Initially, $A' = A$ and $\forall a \in A'$, $C_{r_a} = B_a$ and afterwards, it is updated as follows:

$$C_{r_a} = C_{r_a} - \sum_{f \in F} w(f) * \chi_a$$

with $\chi_a = \begin{cases} 1 & \text{if } a \in sp(f) \\ 0 & \text{otherwise.} \end{cases}$

Let us now define the notions of *admissible assignment* and *admissible fusion*, which for the current approach, verify not only the respect of capacity resources but also the existence of a routing.

Let W be the set of vertices not yet assigned to a node.

Definition 4.4. *An assignment of task t to node n is admissible if it satisfies the capacity constraints for node n:*

$$s_{tr} + \sum_{t' \in V \setminus W : g(t') = n} s_{t'r} \leq C_r, \forall r \in R$$

and there is a feasible routable path for every flow f between t and all the other tasks $t' \in V \setminus W$ with $g(t) \neq g(t')$ and $(tt') \in E$:

$$\{\exists sp(f) \in G' : s(f) = t \wedge d(f) = t' \wedge w(f) = q_{tt'} > 0\}$$

$$\{\exists sp(f) \in G' : s(f) = t' \wedge d(f) = t \wedge w(f) = q_{t't} > 0\}$$

Definition 4.5. *A fusion between the nodes n and m is admissible if:*

$$\sum_{t \in V \setminus W : g(t) = n} s_{tr} + \sum_{t \in V \setminus W : g(t) = m} s_{tr} \leq C_r, \forall r \in R$$

and all the flows for tasks belonging to n and m are reroutable through G'.

After each assignment or fusion, G' and F are updated accordingly, by modifying C_{r_a} and by adding and/or removing flows (in the case of a fusion). The overall framework of the greedy randomized construction algorithm is presented in Alg.4.2. Initially, a partial solution is set as the first $\min(|V|, |N|)$ tasks in lexicographic order assigned to the N nodes with the condition that this initial mapping is also routable.

Then, the list $[rcl]$ of k best decisions is constructed in a greedy fashion, by choosing between an admissible assignment or an admissible fusion, the ones with the highest affinity. Between the parameters we can set before running our algorithm, we can also define which type of affinity (relative or absolute) we want to choose as criterion for deciding between several candidates in the constructive part.

Once a decision c_i is chosen at random from $[rcl]$, we evaluate its nature (assignment or fusion) and make the corresponding changes for C_{r_a} and F.

Algorithm 4.2: GRASP for joint placement and routing: construction_phase

---

Input: V, N, R, k

1: Initialization of the set of unassigned tasks $W = V$
2: Assign the first $\min(|V|, |N|)$ vertices to the $|N|$ nodes and update sets W, F
3: Build the list of k restricted candidate decisions $[rcl]$ made of admissible assignments (cf. Def.4.4) and admissible fusions (cf. Def.4.9)
4: Select at random c_i from $[rcl]$
5: If c_i is an assignment ($v^* \in W, n^* \in N$), then update set W.
Else, c_i is a fusion ($n_1^* \in N, n_2^* \in N$), and thus merge nodes n_1^* and n_2^*.
6: Update the reduced graph G' and set of flows F
7: If $W = \emptyset$ or there is neither any admissible assignment nor any admissible fusion, stop.
Else, go to Step 3.

Output: Assignment $g_c(V)$

If c_i is an assignment of task t_i to node n, the set W is updated: $W = W \setminus \{t_i\}$, the incoming / outgoing flows between the task t_i and the other tasks already assigned are computed and added to the set F and the residual capacities of the arcs of the network are reduced accordingly.

If c_i is a merge of two nodes ($n_1^* \in N, n_2^* \in N$), the necessary modifications are made such that all vertices from node n_1^* are transferred to node n_2^*, the flows of the tasks already assigned are updated for taking into account the fusion and the residual capacities of the arcs of G' are also recomputed.

4.3.3 Local search phase

Afterwards, the quality of the constructed solution S for g_c, the assignment obtained previously, is improved through a local search procedure. The neighborhood structures are classical: either 1-OPT by transferring single tasks already placed to others nodes or 2-OPT, consisting in generating a new solution from S by interchanging pairs of tasks assigned to different nodes. The use of this type of neighborhoods is appropriate under the assumption of a relative homogeneity for the tasks weights.

Also, when setting the parameters of the local optimization we can choose between a first (in which the current solution is replaced by the first better local solution) or best improving search strategy. In practice, it has been observed that for many applications, quite often, both search strategies lead to the same final solution, but with smaller computation times when a first improving strategy is used [63].

The subtlety of our approach consists in selecting the tasks to move and exchange from the set:

$$EX_t = \{t \in V : (\exists n \neq g(t) \in N : \alpha(t,n) - \alpha(t,g(t)) > 0)\}$$

with $\alpha(t,n)$, the affinity of task t for node n (see [144], [149]).

Once the set EX_t is constructed, only *admissible transfers* or *admissible exchanges* are analyzed. The routability aspect is verified using the same principles as described previously and each time a local optimization occurs and the placement is modified, the reduced graph and the set of flows F are also updated.

Definition 4.6. *For a given assignment g, a transfer of task t to a node n is admissible if:*

- *the capacity of node n remains respected for each resource* $\sum_{t_1:g(t_1)=n} s_{t_1 r} + s_{tr} \leq C_{nr},\ \forall r \in R$
- *the flows $f \in F$ between t and other tasks t' for which $g(t') \neq n$ and $w(f) > 0$ are reroutable.*

The solution S' of the new placement when moving t to node n can be easily computed using S: $S' = S + \sum_{g(t')=g(t)} q_{tt'} - \sum_{g(t')=n} q_{tt'}$.

Definition 4.7. *For a given assignment g, an exchange of two tasks t and t' from node $g(t)$ to node $g(t')$ and vice versa is admissible only if:*

- *the capacity constraints for the associated nodes are respected*

$$\sum_{t_1:t_1 \neq t; g(t_1)=g(t)} s_{t_1 r} - s_{tr} + s_{t'r} \leq C_{g(t)r},\ \forall r \in R$$

$$\sum_{t_1:t_1 \neq t'; g(t_1)=g(t')} s_{t_1 r} - s_{t'r} + s_{tr} \leq C_{g(t')r},\ \forall r \in R$$

- *the flows in F having as source or sink t and/or t' are still routable.*

Since, except for the exchanged tasks, all the others remain on the same nodes, the computation of the value for the solution S' corresponding to a 2-OPT neighborhood can be realized quickly based on S and the bandwidths of exchanged tasks. The new value of the solution when moving t to $g(t')$ and t' to $g(t)$ will be:

$$S' = S + \sum_{g(t)=g(t_i)} (q_{tt_i} - q_{t't_i}) + \sum_{g(t')=g(t_i)} (q_{t't_i} - q_{tt_i})$$

4.4 Stochastic algorithm

For the stochastic version of the joint placement and routing, we consider that the random data are the weights of the tasks and we obtain the associated chance-constrained problem, in which constraints 4.3 are being replaced by the probability constraints for the capacities of the nodes:

$$\mathbb{P}\left(\sum_{t\in V:g(t)=n} s_{tr} \leq C_{nr},\ \forall n \in N; \forall r \in R\right) \geq 1-\varepsilon.$$

with $\varepsilon \in (0,1)$.

We also assume that, for the weights of each task $t \in V$, for each resource $r \in R$, we have at our disposal a sample of sufficient size NS of i.i.d. realizations $\tilde{s}_{tr}^{(1)}, \ldots, \tilde{s}_{tr}^{(NS)}$.

As such, in order to solve this stochastic problem, we can use the same methodology as the one described in Chapter 2, Section 2.3.5, and adapt the existing GRASP by integrating the robust binomial approach.

Let us recall that the necessary changes for modifying an algorithm for the deterministic case into one solving the chance-constrained version were at the level of the oracle deciding the admissibility of a solution. A solution is accepted in the stochastic case if the number of times the original constraint is respected is superior to the threshold k established in function of NS, the size of the sample, the initial probability level $1-\varepsilon$ and of the confidence level $1-\alpha$ with $\alpha \in (0,1)$.

For the GRASP conceived for the joint deterministic placement and routing, the oracle of the greedy constructive step which decides if a decision (either assignment or fusion) is feasible is based on the notions of *admissible assignment* and *admissible fusion*. Therefore, we have to modify these two notions in order to take into account the stochastic nature of the tasks weights.

Since the constructive part is inspired from the existing algorithm for graph partitioning which we have also adapted to the stochastic case, under the same assumptions, the notions of *stochastic admissible assignment* and *stochastic admissible fusion* will be similar to those from Chapter 3 with the exception of the routability aspect to be taken into account.

Definition 4.8. *An assignment of task t to node n is stochastically admissible if:*

– *the sum*

$$\sum_{i=1}^{NS} \chi(\{\exists n' \neq n, \exists r : \sum_{t':g(t')=n'} \tilde{s}_{t'r}^{(i)} > C_r\} \vee \{\exists r : \tilde{s}_{tr}^{(i)} + \sum_{t':g(t')=n'} \tilde{s}_{t'r}^{(i)} > C_r\}),$$

is less than $NS - k(NS, 1-\varepsilon, \alpha)$*, where* $\chi(\mathcal{P}_a) = 1$ *if and only if the predicate* $\mathcal{P}_a$ *is true.*

– *there is a feasible routable path for every flow* f *between* t *and all the other tasks* $t' \in V \setminus W$ *with* $g(t) \neq g(t')$ *and* $(tt') \in E$*:*

$$\{\exists sp(f) \in G' : s(f) = t \wedge d(f) = t' \wedge w(f) = q_{tt'} > 0\}$$

$$\{\exists sp(f) \in G' : s(f) = t' \wedge d(f) = t \wedge w(f) = q_{t't} > 0\}$$

Definition 4.9. *A fusion between the nodes* n *and* m *is stochastically admissible if:*

– *the sum*

$$\sum_{i=1}^{NS} \chi(\{\exists n', r : \sum_{t:g(t)=n'} \tilde{s}_{tr}^{(i)} > C_r\} \vee \{\exists r : \sum_{t:g(t)=n} \tilde{s}_{tr}^{(i)} + \sum_{t':g(t')=m} \tilde{s}_{t'r}^{(i)} > C_r\}),$$

is less than $NS - k(NS, 1-\varepsilon, \alpha)$*, where* $\chi(\mathcal{P}_f) = 1$ *if and only if the predicate* $\mathcal{P}_f$ *is true.*

– *all the flows for tasks belonging to* n *and* m *are reroutable through* G'*.*

Therefore, the only major modifications for the greedy algorithm 4.2 are during the step 2 and 3 in which the admissibility criterion are used.

As for the post-optimization step, the local search is based on the notions of *admissible transfer* or *admissible exchange* which are defined with regards to the weights of the tasks and the capacity of each node. Thus, we have to modify these two notions by applying the robust binomial approach.

Definition 4.10. *For a given assignment* g*, a transfer of task* t*, already assigned to node* $n_i = g(t)$*, to another node* n *is stochastically admissible if:*

– *the flows* $f \in F$ *between* t *and other tasks* t' *for which* $g(t') \neq n$ *and* $w(f) > 0$ *are reroutable.*

– *the sum* $\sum_{i=1}^{NS} \chi(\mathcal{P}_a) < NS - k(NS, 1-\varepsilon, \alpha)$ *with*

$$\begin{aligned}\mathcal{P}_a :\{&\{\exists n' \neq n \neq n_i, \exists r : \sum_{t':g(t')=n'} \tilde{s}_{t'r}^{(i)} > C_r\} \\ &\vee \{\exists r : \sum_{t':g(t')=n} \tilde{s}_{t'r}^{(i)} + \tilde{s}_{tr}^{(i)} > C_r\} \\ &\vee \{\exists r : \sum_{t':t'\neq t;g(t')=n_i} \tilde{s}_{t'r}^{(i)} - \tilde{s}_{tr}^{(i)} > C_r\}\}\end{aligned}$$

where $\chi(\mathcal{P}_a) = 1$ *if and only if predicate* $\mathcal{P}_a$ *is true.*

Definition 4.11. *For a given assignment* g*, an exchange of two tasks* t *and* t' *from node* $g(t)$ *to node* $g(t')$ *and vice versa is stochastically admissible if:*

– *the flows in* F *having as source or sink* t *and/or* t' *are still routable.*

– *the sum* $\sum_{i=1}^{NS} \chi(\mathcal{P}_a) < NS - k(NS, 1-\varepsilon, \alpha)$ *with*

$$\mathcal{P}_a : \{\{\exists n' \neq g(t) \neq g(t'), \exists r : \sum_{t_1 : g(t_1)=n'} \tilde{s}^{(i)}_{t_1 r} > C_r\}$$
$$\vee \{\exists r : \sum_{t_1 : t_1 \neq t; g(t_1)=g(t)} \tilde{s}^{(i)}_{t_1 r} + \tilde{s}^{(i)}_{t' r} - \tilde{s}^{(i)}_{tr} > C_r\}$$
$$\vee \{\exists r : \sum_{t_1 : t_1 \neq t'; g(t_1)=g(t')} \tilde{s}^{(i)}_{t_1 r} + \tilde{s}^{(i)}_{tr} - \tilde{s}^{(i)}_{t' r} > C_r\}\}$$

where $\chi(\mathcal{P}_a) = 1$ *if and only if predicate* $\mathcal{P}_a$ *is true.*

Besides these changes when defining the admissible neighborhoods, the local search remains the same as for the deterministic problem.

Let us now provide some experimental results obtained by applying the GRASP method for deterministic and respectively stochastic case.

4.5 Computational results

4.5.1 Benchmarks

4.5.1.1 Deterministic instances

In order to test our GRASP algorithm, we used several sets of test problems: grids to be placed on square grids, a modified version of Johnson instances [86], random data generated with TGFF [1] and a real image processing application to be compiled using the compilation chain and placed on a manycore architecture.

The first set of instances consists of undirected DPNs grids, representative in size for our application context, with unitary weights for tasks and for communication channels. Besides, these instances are easy to modify and we can use them to test different configurations. Table 4.1 shows grids instances details, with column "#Vertices" the number of vertices to be placed and column "#Nodes" the number of clusters for a homogeneous tore architecture on which the vertices have to be placed. The results are giving for a maximal bandwidth for the links of the different NoCs set to $B_a = 1000$. The end column "Sol." reports the solutions obtained by the semi-greedy algorithm for tasks mapping described in [144].

The second set is composed of publicly available undirected graphs, first used for bipartitioning [86], with different topologies and a number of vertices varying between 124 and 1000. We consider unitary weights for the channels between each communicating pair of vertices as well as unitary mono-dimensional weights for the vertices. The initial instances were adapted to be placed on a torus 2D of 4×4 nodes with maximal capacity on the arcs $B_a = 1000$.

The real application we test here is the motion target application, video processing and tracking a sequence of related input video frames as described in Chapter 1. Modifying the number of strips in which the images of the video sequence are divided induces a modification of the number of tasks to be placed. There are three

1. Tasks Graph for Free: http://ziyang.eecs.umich.edu/ dickrp/tgff/

Table 4.1: Grid instances

Inst.	#Vertices	#Nodes	C_n	Sol.
Grid 4×4	16	4	4	8
Grid 10×10	100	16	7	70
Grid 12×12	144	4	40	31
Grid 18×18	324	9	40	88
Grid 23×23	529	16	40	162

kinds of resources for the node capacity: cardinality, computing core occupancy and memory footprint. The application has to be placed on a bi-dimensional torus 4×4.

The random tasks graphs instances generated with TGFF are 1920 graphs with the number of vertices V varying between $\{50, 100, 200\}$ to be placed on a clusterized bi-dimensional architecture with $N = 4$ or $N = 16$ nodes. For each set of graphs composed of 50, 100 and respectively 200 vertices, four seeds are used for generating different communications and occupancy ratios. The number of incoming and outgoing arcs a task can have is limited to two. We considered the mono-resource case in which the capacity constraints are on the occupation ratios of each node. The capacities of nodes $n \in N$ of the architecture are equal and are computed as: $C_n = x * \sum_{i=1}^{V} s_i/N$ with s_i the weight of task i and $x \in \{1.01, 1.25, 1.5, 1.75, 2\}$. As for the maximal bandwidth B_a on the arcs of the target architecture, we create and sort the list of communications weights of the channels between tasks $l = \{q_{t_i t_j} > 0 : t_i, t_j \in V\}$ and then choose B_a as $max(q_{t_i t_j}) + \sum_{i=1}^{y} l[i]$ with $y \in \{5, 6, 7, 8\}$. Therefore, the most restricted instances are those with limited capacity on the nodes when $x = 1.01$ and with limited maximal bandwidth for the arcs of the network when $y = 5$.

4.5.1.2 Stochastic instances

The tests for the chance-constrained version of the placement and routing were performed on the above instances, transformed to stochastic benchmarks with random weights for the tasks.

For the grids instances, we generated the random variables representing the weights of the vertices by simulating a joint bimodal distribution with each mode uniform in its intervals and selected in an equally likely manner. The first mode is represented by the hypercube: $[0.8, 0.9]^{|V|}$, and the second one, by the hypercube: $[1.1, 1.2]^{|V|}$.

As for the TGFF instances, we considered small variations on the weight w_t of each task t, following a bimodal uniform distribution : $[w_t - 3\%w_t, w_t - 1\%w_t] \times [w_t + 1\%w_t, w_t + 3\%w_t]$.

For the target motion detector, we consider the case when this ΣC application is composed of 57 tasks and has to be mapped on a Kalray architecture [56], with a frequency of the chip of 400MHz. We use a simulation with ISS (Instruction Set Simulator) to obtain the processor cycles for each execution of an agent and thus, deducing the execution times(knowing that a cycle corresponds to 2.5 ns).

Instead of computing the core occupancy for each agent based on the mean of these executions (as it is made in the deterministic case), we take a sample of minimum 30 occupation rates of each occurrence for an agent and apply the GRASP for the stochastic case to place the application. Each task is repeated 1 times per execution cycle and the application is dimensioned to get 30 frames per second in output. As such, the occupancy ratio of each occurrence of an agent, having a processor cycle p is calculated as $\%\text{ratio} = \frac{\text{computed rate}}{\text{max rate}} = 2.5 * 30 * p_{(\text{sec})} * 10^2$.

The experiments on grid instances, Johnson benchmark and the image processing application have been carried out on a Linux workstation, with a 2.40 GHz I5 processor, 8 GB of memory and Ubuntu 12.04 as operating system. The benchmark composed of TGFF graphs has been tested on a Linux workstation, with 48 processors, 64 GB of memory and Ubuntu 12.04 as operating system.

4.5.2 Results for the deterministic version

Our GRASP algorithm was tested for different configurations, with $k \in \{2, 3, 4\}$ (see Alg. 4.2, line 3), total versus relative affinity during construction, best improvement and first improvement, 1-OPT versus 2-OPT for local search phase, etc.

We have decided to stop our algorithm when a number of maximal iterations or when a time limit of 10 minutes are reached.

Since we prioritize the minimization of the bandwidths (cf. Eq.4.1) we guarantee just that this mapping is routable. As such, we are not guaranteeing an optimal routing and instead, we are analyzing the difference, for an obtained placement, between the routing our algorithm is using and an ideal one, (using a shortest-path strategy), by measuring the average for all flows $f \in F$ of fraction: $lb = \frac{length(sp(f))}{length^r}$ with $length^r$ being the shortest path in the NoC between $s(f)$ and $d(f)$.

Table 4.2 shows some of the placement results obtained for grids instances when the number of iterations is equal to $\max(100, |V|log|V|)$, the notion of relative affinity is used, the maximal bandwidth $B_a = 1000$ and the number of selections $k \in \{2, 3, 4\}$. The column "GR" represents the results of the construction part while columns "PS-1" and "PS-2" are the complete results with post-optimization, when 1-OPT and respectively 2-OPT neighborhoods are used. As shown, the local search is useful and better results are obtained for $k = 2$ and $k = 3$. Overall the quality of solutions is comparable with the one found by the algorithm from [144]. GRASP solutions have an average deviation from the solutions found by the semi-greedy method in [144] of ≈5% for $k = 2$ (with 2-OPT) and less than 10% for $k = 3$ and $k = 4$ (both 1-OPT and 2-OPT), with the advantage that we also ensure the routability. In average, the results found using 2-OPT are better than those with 1-OPT. When the capacity of arcs B_a is large enough, our method is able to accommodate the flows via the shortest paths and $lb = 1$ in all cases. Instead, when limiting more the capacity of the links, the average of lb tends to increase to 1.05.

For the second set, as shown in Table 4.3 the best values for the placement of our GRASP were obtained with the notion of relative affinity, when $k = 2$ and $k = 3$ with solutions of better quality than those found by [144], (reported in columns

Table 4.2: Results of GRASP method for grid problems

	k=2			k=3			k=4		
Name	GR	PS-1	PS-2	GR	PS-1	PS-2	GR	PS-1	PS-2
Grid4x4.inst	11	11	10	12	10	11	13	12	10
Grid10x10.inst	73	69	69	75	70	69	76	69	69
Grid12x12.inst	34	31	30	34	31	30	36	31	33
Grid18x18.inst	92	86	88	91	91	91	99	98	91
Grid23x23.inst	174	164	173	184	173	177	190	184	182

"Greedy") for 18 and respectively 17 instances (out of a total of 25). Also, the results are definitely better for $k = 2$ instead of $k = 4$ (for 20 out of 25 instances).

For the target motion application, Table 4.4 shows the results obtained for a number of processes varying between 60 and 300 (column "$|V|$") in function of the number of strips (column "ST"). These results, obtained with the GRASP approach for $k \in \{2, 3, 4\}$, using the notion of total affinity and a number of iterations equal to $\max(100, |V|log|V|)$, are compared with those obtained by the method currently implemented in the compilation chain (column "[144]") for the placement of the application on a 2D torus 4×4 with $B_a = 10000000$. The GRASP method provides better results in almost all cases. It should however be noted that when relative affinity is used instead, the results of the GRASP are of lower quality. Since the capacity of the network is large enough with regard to the flows to be routed, the bound lb is equal to 1 for all instances, meaning that the routes found are following shortest paths.

The same instances were used to place the target motion application on the same homogeneous NoC but this time with a maximal bandwidth for each arc $B_a = 100000$. While none of the placements found by the method from [144] is routable afterwards, the current method is finding placements which are also routable, with an average of 1.17 for lb.

Extensive tests were also performed on the random TGFF graphs. One of the first tests was to compare the quality of the solutions for a different number of maximal iterations, when $k = 2$, relative affinity is used and local search is based on exchanges of tasks. As expected, more the number of iterations is higher, more the quality of solutions increases, with $\approx$50% of cases in which the solutions are better for $\max(100, |V|log|V|)$ iterations.

We then compared the quality of the placements for a number of selections equal to 2 and post optimization based on 2-OPT, when the notions of total and relative affinity are used. It seems that the relative affinity is a better criterion to choose for the construction part, with 1113 instances with solutions of higher quality against 268 when using the absolute affinity.

Another test consisted in testing the GRASP (with $k = 2$, the number of iterations $\max(100, |V|log|V|)$ and a 2-OPT strategy) against the sequential algorithm from [144]. The last one solves first the placement with a greedy method and afterwards the routing with a MILP. The GRASP is able to find more solutions (for a total of 1920 instances), with 1358 instances against 927 for the algorithm in [144].

Table 4.3: Results of GRASP method with $\max(100, |V|log|V|)$ iterations for Johnson instances compared with greedy method from [144]

Name	$\|V\|$	C_n	GRASP			[144]
			k=2	k=3	k=4	
G.sub.500	500	33	602	605	603	597
G1000.0025	1000	63	331	331	339	336
G1000.005	1000	63	1262	1259	1266	1248
G1000.01	1000	63	3333	3335	3335	3376
G1000.02	1000	63	7632	7631	7654	7676
G124.02	124	8	53	54	53	52
G124.04	124	8	183	183	183	187
G124.08	124	8	446	445	445	446
G124.16	124	8	1025	1025	1025	1029
G250.01	250	16	105	106	106	103
G250.02	250	16	325	327	327	330
G250.04	250	16	870	874	877	884
G250.08	250	16	1860	1865	1872	1881
G500.005	500	33	173	172	175	167
G500.01	500	33	624	627	627	637
G500.02	500	33	1543	1543	1550	1562
G500.04	500	33	3893	3909	3927	3922
U1000.05	1000	63	99	110	125	117
U1000.10	1000	63	467	469	518	514
U1000.20	1000	63	1642	1675	1780	1700
U1000.40	1000	63	5267	5096	5318	5308
U500.05	500	33	96	94	98	87
U500.10	500	33	335	371	358	353
U500.20	500	33	1132	1118	1144	1188
U500.40	500	33	3667	3653	3625	3610

Table 4.4: Results for target motion application compared with greedy method from [144]

Name	ST	$\|V\|$	GRASP			[144]
			k=2	k=3	k=4	
MD1.in	8	67	538206	538206	538206	538206
MD2.in	10	81	492530	492530	492530	492536
MD3.in	15	116	492934	492934	492934	492944
MD4.in	20	151	511701	511701	541620	496353
MD5.in	30	221	525268	525269	515030	535525
MD6.in	40	291	542059	541661	526507	587142

For 25,5% of the total number of graphs, our algorithm finds a routable placement while the other does not find a placement or finds a placement which is not routable.

When both algorithms find a solution, the value of the placement of the GRASP is better or within 5% of the value found by the other method for 28.7% of cases. For the routing, in 38.3% of solved cases, the values are within 7% from the optimal routing found by the MILP from the sequential algorithm.

4.5.3 Results for the stochastic version

The tests for the stochastic version of the placement and routing are performed with the following configuration for the GRASP: the number of selections $k = 2$, relative affinity as criterion of choice in the constructive part, local search based on 2-OPT and the maximal number of iterations fixed to $\max(100, |V|log|V|)$. We decided to stop the algorithm as before, when the maximal number of iterations is reached or a time limit of 10 minutes are reached.

The experiments consist in evaluating aspects such as the quality of the placements, the time required and "the price of robustness". First we keep the same capacity for all nodes as in the deterministic case and afterwards, if needed, increase the capacity of all nodes with a factor of $\{1.15, 1.25, 1.5, 1.75\}$ until a feasible solution for the chance-constrained case is found.

The stochastic version of the GRASP was tested on the grids problems by varying the parameters $\varepsilon \in \{0.05, 0.1\}$ and α in $\{0.01, 0.05\}$ for a sample size of 100 and respectively 1000. Tables 4.5-4.6 report the solutions obtained with column "sol." for the solution value, columns "time" for the execution time and "C_n" the increase factor required for the capacity of each node in order to find a feasible solution.

Table 4.5: Computation results for $NS = 100$: grid problems

instance	α	$\varepsilon = 0.05$			$\varepsilon = 0.1$		
		sol.	C_n	time	sol.	C_n	time
grid4×4	0.01	10	1.25	≈0	12	1.25	≈0
	0.05	10	1.25	≈0	12	1.25	≈0
grid10×10	0.01	74	1,15	0,92	76	1,15	0,56
	0.05	72	1,15	0,52	69	1,15	0,48
grid12×12	0.01	30	1	77	28	1	77,68
	0.05	32	1	58,5	28	1	67
grid18×18	0.01	92	1	600	89	1	600
	0.05	94	1	600	94	1	600

As it can be seen, the quality of the solutions is coherent with those found by the deterministic algorithm. Also, we can remark that the effort to achieve robustness for the solutions is not so high. For instances "grid4×4" and "grid10×10" it is necessary an increase of 1.25 and respectively 1.15 in the capacity of the node in order to find a solution. For the other instances, the stochastic GRASP is able to find solutions by keeping the same C_n. As expected, the execution time of the

Table 4.6: Computation results for $NS = 1000$: grid problems

instance	α	$\varepsilon = 0.05$			$\varepsilon = 0.1$		
		sol.	C_n	time	sol.	C_n	time
grid4×4	0.01	10	1,25	≈0	12	1,25	≈0
	0.05	10	1,25	≈0	8	1,25	≈0
grid10×10	0.01	75	1,15	0,69	78	1,15	0,79
	0.05	76	1,15	0,72	79	1,15	0,95
grid12×12	0.01	29	1	230	31	1	210,6
	0.05	30	1	235,3	29	1	263
grid18×18	0.01	95	1	600	94	1	600
	0.05	91	1	600	93	1	600

method depends on the number of vertices and on the size of the sample, with a superior overall execution time when using a sample size of 1000 instead of a sample of 100 realizations.

We also tested the algorithm on the 1920 stochastic TGFF instances for a sample size of 100, when $\varepsilon = 0.95$ and $\alpha = 0.05$. Table 4.7 shows the average time needed to find solutions for sets of instances having same number of vertices V: 50, 100 and respectively 200 and confirms our assessment on the computational complexity increasing with the number of vertices.

Table 4.7: Computation time for $NS = 100$, $\varepsilon = 0.05$, $\alpha = 0.05$: TGFF problems

	V		
	50	100	200
#instances	640	640	640
Time (sec.)	16,91	140,69	486,68

As reported in Table 4.8, the majority of robust solutions (68.33%) are found without the need to increase the capacity of each node C_n (column "1"). While in ≈14% of cases a multiplication factor of 1.15 for C_n is required to reach probabilistic solutions (usually for initial instances with limited node capacity), for 12.4% of instances, our method is unable to find solutions (column "NA"). We can remark however that for the last category, the initial deterministic GRASP also has not found solutions and only 11 additional instances are not solved for the chance-constrained version. Moreover, the stochastic method finds more feasible solutions than its deterministic counterpart, since it is more flexible by allowing the increase of the node capacity.

We have also compared the quality of the solutions with those found in the deterministic case when the capacity of the node remains the same. The results are synthesized in Table 4.9 where the value of the stochastic solution sol_s is compared to the deterministic solution sol_d. For more than 40% of the 1312 instances, the solutions obtained are of better quality than in the deterministic case and in more

Table 4.8: Repartition of solutions for $NS = 100$, $\varepsilon = 0.05$, $\alpha = 0.05$ in function of C_n: TGFF problems

	Multiplication factor for C_n				
	1	1.15	1.25	1.5	1.75
% instances	68,33	13,91	1,46	1,09	2,81

than 38% of cases, the value of the stochastic solution is at most 5% different from the one of the deterministic instance.

Table 4.9: Quality of stochastic vs. deterministic solutions for same C_n: TGFF problems

	%instances	
sol_s	$< sol_d$	41.92
	$\{sol_d; sol_d + 5\%\}$	38.87
	$\{sol_d + 5\%; sol_d + 7\%\}$	5.11
	$\{sol_d + 7\%; sol_d + 10\%\}$	4.88
	Other	9.22

Finally, we tested the stochastic algorithm on real data obtained by running the motion target application on ISS simulator with four different inputs, $\varepsilon = 0.1$ and $\alpha = 0.05$. The size of the samples for the computation ratios of the tasks as well as the values of the placements and routing are reported in Table 4.10. Each time we obtain same results for placement and for routing as the GRASP for the deterministic case and the sequential algorithm from [144]. One possible explanation is the small sizes of the instances and the reduced quantity of resources they require: for each input, only two out of the 16 clusters are used to map the application.

Table 4.10: Results stochastic GRASP for $\varepsilon = 0.1$ and $\alpha = 0.05$: motion target application

Inst.	#Vertices	#Nodes	NS	Sol. placement	Sol. routing
scenario1.list	57	16	33	16480	16480
scenario2.list	57	16	85	16512	16512
scenario3.list	57	16	35	16504	16504
scenario4.list	57	16	85	16552	16552

Conclusion

In this chapter we addressed the problem of joint placement and routing of dataflow applications on a clusterized architecture, for both deterministic and stochas-

tic cases. In order to find routable placements, we have designed a GRASP for the deterministic version which was further adapter for the stochastic case using the robust binomial approach introduced in Chapter 2. For each assignment of a task to a cluster or a change on the current mapping, the routability is verified via a shortest-path algorithm on a residual graph, build from the initial architecture and updated constantly.

Extensive experiments were performed either on random generated instances or on real data obtained for the motion target application. When tested on a benchmark composed of 1920 synthetic graphs, for 25% of cases, our GRASP method found mappings which are also routable while a sequential algorithm (doing first the placement and the routing afterwards) did not find any valid solutions. Also, for the benchmark consisting of motion target data, for a reduced maximal available bandwidth on the arcs of the network, our algorithm is able to find routable placements of good quality. The heuristic method for the deterministic problem and some preliminary results are introduced in [151].

As for the stochastic problem, the GRASP is able to find solutions of good quality without paying too much of a price to obtain robustness. The tests run on grids, random graphs and motion target data showed that taking into account the variations of the data is particularly important in cases when the available resources are limited.

With the arrival of new embedded applications, more complex to deal with and more computationally demanding, we feel that the joint placement and routing can be applied as a possible alternative to map these programs. Moreover, for situations where the clusters resources become tight, solving the mapping by considering uncertainty may prove to be useful.

Conclusion and future work

The new generation of manycore embedded systems, containing hundreds if not thousands of cores, requires new programming and execution models for parallel applications in order to fully take advantage of the available computing power.

Dataflow paradigm seems a good solution to program applications for these manycore architectures which can overcome the associated difficulties (limited and dependent resources, parallelism, etc). However, in order to deploy dataflow applications on the target platforms and efficiently exploit the resources, one must resort to optimization techniques from the operations research field all along the compilation process. Additionally, one common particularity of the optimization problems related to this domain is the presence of uncertain data (such as execution times or network latencies).

In this thesis, we have treated optimization under uncertainty in the context of massively parallel embedded systems. The overall purpose was to apply operation research techniques in order to solve optimization problems from the compilation of dataflow programs for manycores when the data are uncertain.

Taking into account the specificities (dependency, multidimensionality etc.) of execution times, one of major sources of uncertainty for the manycore context, we have conceived a new method for solving chance constrained programs which can be applied without any particular assumption on the random variables. Based on the scenario optimization method, known to be easily applied, and on basic statistic tools, the robust binomial approach is extensible to numerous other application domains. The only requirement is to have at our disposal a sufficiently representative sample of observations. The approach is truly algorithmic efficient if we make use of it within the framework of approximate algorithms or of heuristics, when deciding the feasibility of a solution.

A general methodology has been designed for adapting existing (meta)heuristics to solve the stochastic problems by integrating the robust binomial approach. In this way, complex problems can be tackled in order to find robust solutions, guaranteed with a minimal reliability threshold and with a high confidence level. Moreover, extending an algorithm already developed to solve the stochastic case of a same problem is relatively easy in terms of software engineering. Therefore, applying (meta)heuristics enhanced with the robust binomial approach may be a flexible and viable alternative to address real-world size chance constrained problems.

The robust binomial approach was validated while studying two optimization problems from the compilation of embedded dataflow applications: the partitioning as well as the joint placement and routing of networks of processes.

For the first problem, consisting of finding a partition of processes onto a fixed number of nodes when the processes have uncertain weights, we have proposed a greedy resolution method.

The second problem having as objective the minimal assignment of the processes which is also routable, was treated in both deterministic and stochastic cases. A GRASP heuristic was first developed for the deterministic version and afterwards adapted to solve the stochastic case with variations on the weights of the processes.

For each problem, the quality of the solutions found by the resolution methods proposed has been established with experimental tests on synthetic benchmarks and even on practical instances (a motion target application).

Clearly, many other research directions remain to be explored, related to either the general optimization under uncertainty methodology or to the application context.

With regards to the robust binomial approach, it would be interesting to find new ways to improve it. An example would consist in finding a method for classifying the set of observations from the initial sample in different groups, targeting different probability levels for the constraints. Since our method focuses on finding feasible but suboptimal solutions to chance constrained programs, one of the area needing further investigation concerns the quality of the proposed solutions. As such, one appealing direction of research would be to find specific contexts for which the heuristics integrating the robust binomial approach provide high quality approximation algorithms or for which theoretical upper bounds can be established. Also, another open question for a given chance constrained problem is the choice of the "best" metaheuristic to be applied.

Overall, we feel the need for further approaches for optimization under uncertainty developed from a data-driven perspective. While many real-world domains are characterized by huge and rich amounts of data, most existing models from stochastic optimization literature miss a direct connection with the data! Consequently, we consider treatment, analysis and exploration of experimental data as a prerequisite in designing techniques of optimization under uncertainty appealing in a practical sense. A first step has been taken in this direction by the study of mixtures of Uniform and Gaussian distributions since there can be particular cases in which random data can be associated to such distributions. Estimation of parameters for these laws from a given sample, equivalent to a combinatorial optimization problem, can be a preceding step for resolution of chance constrained program which could take further advantage of it.

With regards to the application studies we covered, there are also subject to further investigations. For instance, for the stochastic partitioning of process networks, it is worth considering working on series-parallel graphs, which are similar in structure to the dataflow application we have to deal with. Regarding the joint placement and routing problem, the GRASP method could be improved by the development of a more powerful local search algorithm based on cyclic exchanges of tasks. Additionally, the study of stochastic problem could be completed by taking into account the uncertainty on the inter-tasks bandwidths.

It should be emphasized that the problems we dealt with are close to the execution model for a real-world manycore architecture. Therefore, the resolution

algorithms we conceived could be considered for a future integration into ΣC compilation chain, the result of a collaboration between CEA List laboratory [1] and Kalray, a semiconductor industry partner [2]. Furthermore, they target a broad spectrum of multimedia applications from video encoding standards to motion targeting application or cognitive radio.

Of course, parallel implementations could facilitate the integration of heuristics based on the robust binomial approach and, in general, of optimization techniques to cope with uncertainty, into the compilation process for embedded manycore. As a consequence, another interesting direction of research is to find the most appropriate methods for parallelizing such algorithms, making efficient use of the resources offered by modern workstations (multi-cores, GPU of graphic cards etc.).

Furthermore, due to the intrinsic presence of uncertain data all along the compilation chain, other combinatorial problems than those we studied can be tackled from the stochastic point of view. Finally, we hope that our contribution is a good starting point for applying stochastic optimization to embedded manycores and that, in the future of this emerging field, operations research techniques for dealing with uncertainty will become current practice.

1. http://www-list.cea.fr/
2. http://www.kalray.eu/

Bibliography

[1] A. Agresti and B. A. Coull. Approximate Is Better than "Exact" for Interval Estimation of Binomial Proportions. *The American Statistician*, 52(2) :119–126, 1998.

[2] S. Amarasinghe, M.I. Gordon, M. Karczmarek, J. Lin, D. Maze, R. M. Rabbah, and W. Thies. Language and compiler design for streaming applications. *Int. J. Parallel Program.*, 33(2) :261–278, 2005.

[3] R. Aringhieri. Solving chance-constrained programs combining tabu search and simulation. volume 3059 of *Lecture notes in computer science*, pages 30–41, Berlin, Germany, 2004. Springer.

[4] E. A. Ashcroft and W. W. Wadge. Lucid, a nonprocedural language with iteration. *Commun. ACM*, 20(7) :519–526, July 1977.

[5] P. Aubry, P. E. Beaucamps, F. Blanc, B. Bodin, S. Carpov, L. Cudennec, V. David, P. Dore, P. Dubrulle, B. de D. Dupont, F. Galea, T. Goubier, M. Harrand, S. Jones, J.D. Lesage, S. Louise, N.M. Chaisemartin, T.H. Nguyen, H. Raynaud, and R. Sirdey. 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*, pages 1624–1633, 2013.

[6] R. E. Bellman and L. A. Zadeh. Decision-making in a fuzzy environment. *Management Science*, 17(4) :B–141–B–164, 1970.

[7] W. Ben-Ameur and H. Kerivin. Routing of uncertain traffic demands. *Optimization and Engineering*, 6(3) :283–313, 2005.

[8] W. Ben-Ameur and M. Zotkiewicz. Robust routing and optimal partitioning of a traffic demand polytope. *International Transactions in Operational Research*, 18(3) :307–333, 2011.

[9] A. Ben-Tal and A. Nemirovski. Robust solutions of linear programming problems contaminated with uncertain data. *Mathematical Programming*, 88 :411–424, 2000.

[10] A. Ben-Tal and A. Nemirovski. Robust optimization : Methodology and applications, 2002.

[11] A. Ben-Tal and A. Nemirovski. On safe tractable approximations of chance-constrained linear matrix inequalities. *Mathematics of Operations Research*, 34 :1–25, February 2009.

[12] P. Beraldi and A. Ruszczynski. Beam search heuristic to solve stochastic integer problems under probabilistic constraints. *European Journal of Operational Research*, 167(1) :35–47, 2005.

[13] D. Bertsimas and O. Nohadani. Robust optimization with simulated annealing. *J. of Global Optimization*, 48 :323–334, October 2010.

[14] D. Bertsimas and M. Sim. The price of robustness. *Operations Research*, 52(1) :35–53, 2004.

[15] L. Bianchi, M. Dorigo, L. Gambardella, and W. Gutjahr. A survey on metaheuristics for stochastic combinatorial optimization. *Natural Computing*, 2006.

[16] L. Bic, G. R. Gao, and J.L. Gaudiot. *Advanced Topics in Dataflow Computing and Multithreading*. IEEE Computer Society Press, Los Alamitos, CA, USA, 1995.

[17] C.-H. Bichot. A new method, the fusion fission, for the relaxed graph partitioning problem and comparisons with some multilevel algorithms. *Journal of Mathematical Modelling and Algorithms*, 6(3) :319–344, 2007.

[18] C.-H. Bichot and N. Durand. *Partitionnement de graphe*. Lavoisier, 2010.

[19] G. Bilsen, M. Engels, R. Lauwereins, and J. Peperstraete. Cycle-static dataflow. *IEEE Transactions on Signal Processing*, 44(2) :397–408, February 1996.

[20] G. Bilsen, M. Engels, R. Lauwereins, and J.A. Peperstraete. Cyclo-static data flow. In *1995 International Conference on Acoustics, Speech, and Signal Processing, 1995. (ICASSP-95)*, volume 5, pages 3255–3258, 1995.

[21] T. Bjerregaard and S. Mahadevan. A survey of research and practices of Network-on-chip. *ACM Computing Surveys*, 38(1) :1, 2006.

[22] A. Bonfietti, L. Benini, M. Lombardi, and M. Milano. An efficient and complete approach for throughput-maximal sdf allocation and scheduling on multi-core platforms. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '10, pages 897–902. European Design and Automation Association, 2010.

[23] P. Boulet. Array-OL Revisited, Multidimensional Intensive Signal Processing Specification. Technical Report RR-6113, INRIA, 2007.

[24] L. D. Brown, T. T. Cai, and A. Dasgupta. Interval Estimation for a Binomial Proportion. *Statistical Science*, 16 :101–133, 2001.

[25] I. Buck. Brook Specification v0.2. Technical Report RR-6113, 2003.

[26] J. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt. Readings in hardware/software co-design. chapter Ptolemy : a framework for simulating and prototyping heterogeneous systems, pages 527–543. Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[27] J.T. Buck and E.A. Lee. Scheduling dynamic dataflow graphs with bounded memory using the token flow model. In *IEEE International Conference on Acoustics, Speech, and Signal Processing, 1993. ICASSP-93., 1993*, volume 1, pages 429 –432, april 1993.

[28] A Burns, G. Bernat, and I. Broster. A probabilistic framework for schedulability analysis. In *Proceedings of the Third International Conference on Embedded Software (EMSOFT 2003)*, pages 1–15, 2003.

[29] G. Calafiore and M.C. Campi. Uncertain convex programs : Randomized solutions and confidence levels. *Mathematical Programming*, 102 :25–46, 2005.

[30] G.C. Calafiore and M.C. Campi. The scenario approach to robust control design. *IEEE Transactions on Automatic Control*, 51(5) :742 – 753, may 2006.

[31] M.C. Campi and S. Garatti. A sampling-and-discarding approach to chance-constrained optimization : Feasibility and optimality. *J. Optimization Theory and Applications*, 148(2) :257–280, 2011.

[32] Z. Caner Taskin, J. Cole Smith, S. Ahmed, and A.J. Schaefer. Cutting plane algorithms for solving a stochastic edge-partition problem. *Discrete Optimization*, 6(4) :420 – 435, 2009.

[33] S. Carpov. *Scheduling for memory management and prefetch in embedded multi-core architectures.* Phd thesis, CEA List, Laboratoire Heudiasyc, UMR CNRS 6599, Université de Technologie de Compiègne, 2011.

[34] S. Carpov, L. Cudennec, and R. Sirdey. Throughput constrained parallelism reduction in cyclo-static dataflow applications. *Procedia Computer Science*, 18(0) :30–39, 2013.

[35] S. Carpov, R. Sirdey, J. Carlier, and D. Nace. Memory bandwidth-constrained parallelism dimensioning for embedded many-core microprocessors. In *CPAIOR10 workshop on Combinatorial Optimization for Embedded System Design, Bologna, Italy,*, 2010.

[36] J. Castrillon, A. Tretter, R. Leupers, and G. Ascheid. Communication-aware mapping of KPN applications onto heterogeneous mpsocs. In *Proceedings of the 49th Annual Design Automation Conference*, DAC '12, pages 1266–1271, New York, NY, USA, 2012. ACM.

[37] A. Charnes, W. W. Cooper, and G. H. Symonds. Cost horizons and certainty equivalents : An approach to stochastic programming of heating oil. *Management Science*, 4(3) :pp. 235–263, 1958.

[38] A. Charnes and W.W. Cooper. Chance-constrained programming. *Management Science*, 6 :73–89, 1959.

[39] X. Chen, M. Sim, and P. Sun. A robust optimization perspective on stochastic programming. *Operations Research*, 55(6) :1058–1071, 2007.

[40] J. Choi, H. Oh, S. Kim, and S. Ha. Executing synchronous dataflow graphs on a SPM-based multicore architecture. In *Proceedings of the 49th Annual Design Automation Conference*, DAC '12, pages 664–671, New York, NY, USA, 2012. ACM.

[41] C. J. Clopper and E. S. Pearson. The Use of Confidence or Fiducial Limits Illustrated in the Case of the Binomial. *Biometrika*, 26(4) :404–413, 1934.

[42] A. Colin and I. Puaut. Worst case execution time analysis for a processor with branch prediction. *Real-Time Systems*, 18(2) :249–274, May 2000.

[43] H. Corporaal. 16th International Workshop on Software and Compilers for Embedded Systems, 2013. http ://www.scopesconf.org/scopes-13/.

[44] D.R. Cox and D.V. Hinkley. *Theoretical Statistics*. Chapman & Hall, 1979.

[45] L. Cudennec and R. Sirdey. Parallelism reduction based on pattern substitution in dataflow oriented programming languages. *Procedia CS*, 9 :146–155, 2012.

[46] G.B. Dantzig. Linear programming under uncertainty. *Management Science*, 1(3-4) :197–206, 1955.

[47] V. David, C. Fraboul, J.-Y. Rousselot, and P. Siron. Etude et realisation d'une architecture modulaire et reconfigurable : Projet MODULOR. Technical report, 1/3364/DERI.ONERA, 1991.

[48] D.P. de Farias and B. Van Roy. On constraint sampling in the linear programming approach to approximate linear programming. In *Proceedings of the 42nd IEEE Conference on Decision and Control*, volume 3, pages 2441 – 2446, dec. 2003.

[49] P. de Oliveira Castro. *Expression et optimisation des réorganisations de données dans du parallélisme de flots.* PhD thesis, Université de Versailles Saint Quentin en Yvelines, 2010.

[50] M. Demange and V. Paschos. On an approximation measure founded on the links between optimization and polynomial approximation theory. *Theoretical Computer Science*, 158 :117–141, 1996.

[51] D. Dentcheva, A. Prékopa, and A. Ruszczynski. Concavity and efficient points of discrete distributions in probabilistic programming. *Mathematical Programming*, 89 :55–77, 2000.

[52] D. Dentcheva, A. Prékopa, and A. Ruszczynski. Bounds for probabilistic integer programming problems. *Discrete Applied Mathematics*, 124(1-3) :55 – 65, 2002.

[53] M.V. Devarakonda and R.K. Iyer. Predictability of process resource usage : a measurement-based study on unix. *Software Engineering, IEEE Transactions on*, 15(12) :1579 –1586, 1989.

[54] J.L. Diaz, D.F. Garcia, K. Kanghee, L. Chang-Gun, L. Lo Bello, J.M. Lopez, M. Sang Lyul, and O. Mirabella. Stochastic analysis of periodic real-time systems. In *Real-Time Systems Symposium (RTSS 2002)*, pages 289 – 300, 2002.

[55] R. Diekmann, B. Monien, and R. Preis. Load balancing strategies for distributed memory machines. In *Multi-Scale Phenomena and Their Simulation*, pages 255–266. World Scientific, 1997.

[56] B. Dupont de Dinechin, G. Guironnet de Massas, G. Lager, C. Léger, B. Orgogozo, J. Reybert, and T. Strudel. A Distributed Run-Time Environment for the Kalray MPPA-256 Integrated Manycore Processor. *Procedia Computer Science*, 18(0) :1654 – 1663, 2013.

[57] B. Efron and R.J. Tibshirani. *An introduction to the Bootstrap.* 1994.

[58] L. El Ghaoui, F. Oustry, and H. Lebret. Robust solutions to uncertain semidefinite programs. *Siam J. Optimization*, 9(1) :33–52, 1998.

[59] U. Elsner. Graph partitioning - a survey. Technical report, TU Chemnitz SFB393/97-27, 1997.

[60] F. Ercal, J. Ramanujam, and P. Sadayappan. Task allocation onto a hypercube by recursive mincut bipartitioning. In *Proceedings of the third conference on Hypercube concurrent computers and applications : Architecture, software, computer systems, and general issues - Volume 1*, C3P, pages 210–221, 1988.

[61] N. Fan and P.M. Pardalos. Robust optimization of graph partitioning and critical node detection in analyzing networks. In *Proceedings of the 4th Annual International Conference on Combinatorial Optimization and Applications (COCOA 2010)*, pages 170–183, 2010.

[62] N. Fan, Q.P. Zheng, and P.M. Pardalos. On the two-stage stochastic graph partitioning problem. In *Proceedings of the 5th Annual International Conference on Combinatorial Optimization and Applications (COCOA 2011)*, pages 500–509, 2011.

[63] T.A. Feo and M.G.C. Resende. Greedy randomized adaptive search procedures (GRASP). *Journal of Global Optimization*, 6 :109–133, 1995.

[64] C. E. Ferreira, A. Martin, C.C. de Souza, R. Weismantel, and L.A. Wolsey. The node capacitated graph partitioning problem : A computational study. *Mathematical Programming*, 81 :229–256, 1998.

[65] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *Proceedings of the 19th Design Automation Conference*, DAC '82, pages 175–181, Piscataway, NJ, USA, 1982. IEEE Press.

[66] P-O. Fjällström. Algorithms for graph partitioning : A survey. *Linköping Electronic Articles in Computer and Information Science*, 3, 1998.

[67] Michael J. Flynn. Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, C-21(9) :948–960, September 1972.

[68] R. F. Freund. Optimal selection theory for superconcurrency. In *Proceedings of the 1989 ACM/IEEE conference on Supercomputing*, Supercomputing '89, pages 699–703, New York, NY, USA, 1989. ACM.

[69] A. Gaivoronski, A. Lisser, R. Lopez, and H. Xu. Knapsack problem with probability constraints. *Journal of Global Optimization*, 49 :397–413, 2011.

[70] F. Galea and R. Sirdey. Méthode de cadencement d'applications flot de données cyclostatiques. Technical report, CEA LIST/DACLE/10-070, 2010.

[71] F. Galea and R. Sirdey. A parallel simulated annealing approach for the mapping of large process networks. In *IPDPS Workshop*, pages 1787–1792, 2012.

[72] M.R. Garey, D.S. Johnson, and L Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1(3) :237–267, 1976.

[73] A. Girault, B. Lee, and E.A. Lee. Hierarchical finite state machines with multiple concurrency models. *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems*, 18 :742–760, 1999.

[74] T. Goubier, R. Sirdey, S. Louise, and V. David. ΣC : a programming model and langage for embedded manycores. In *Lecture Notes in Computer Science*, volume 7016, pages 385–394, 2011.

[75] J.L. Gustafson. Reevaluating Amdahl's law. *Commun. ACM*, 31(5) :532–533, May 1988.

[76] C. Hanen and A Munier. Cyclic scheduling on parallel processors : An overview. In Philippe Chrétienne, Edward G. Coffman, Jan Karel Lenstra, and Zhen Liu, editors, *Scheduling theory and its applications*. J. Wiley and sons, 1994.

[77] C.A. Healy, D.B. Whalley, and M.G. Harmon. Integrating the timing analysis of pipelining and instruction caching. In *Proceedings of the 16th IEEE of Real-Time Systems Symposium*, pages 288 –297, dec 1995.

[78] B. Hendrickson and R. Leland. A multilevel algorithm for partitioning graphs. In *Proceedings of the 1995 ACM/IEEE conference on Supercomputing (CDROM)*, Supercomputing '95, New York, NY, USA, 1995. ACM.

[79] R. Henrion and C. Strugarek. Convexity of chance constraints with independent random variables. *Comput. Optim. Appl.*, 41(2) :263–276, November 2008.

[80] L.J. Hong, Y. Yang, and L. Zhang. Sequential convex approximations to joint chance constrained programs : A Monte Carlo Approach. *Operations Research*, 59(3) :617–630, 2011.

[81] J. Hu and R. Marculescu. Energy- and performance-aware mapping for regular NoC architectures. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 24(4) :551–562, 2005.

[82] R. Iannucci et al. *Multithreaded computer architecture : a summary of the state of the art*, volume SECS 0281 of *The Kluwer international series in engineering and computer science*. 1994.

[83] M.A. Iverson, F. Ozguner, and G.J. Follen. Run-time statistical estimation of task execution times for heterogeneous distributed computing. In *Proceedings of 5th IEEE International Symposium on High Performance Distributed Computing, 1996.,*, pages 263 –270, aug. 1996.

[84] A. Jantsch and I. Sander. Models of computation and languages for embedded system design, 2005.

[85] D.S. Johnson, C.R. Aragon, L.A. McGeoch, and C. Schevon. Optimization by simulated annealing : An experimental evaluation ; part i, graph partitioning. *Operations Research*, 37(6) :865–892, 1989.

[86] E.J.L. Johnson, A. Mehrotra, and G. L. Nemhauser. Min-cut clustering. *Mathematical Programming*, 62 :133–151, October 1993.

[87] G. Kahn. The semantics of simple language for parallel programming. In *IFIP Congress*, pages 471–475, 1974.

[88] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20 :359–392, 1998.

[89] S. Kataoka. A stochastic programming model. *Econometrica*, 31(1/2) :181–196, 1963.

[90] B.W. Kernighan and S. Lin. An Efficient Heuristic Procedure for Partitioning Graphs. *The Bell system technical journal*, 49(1) :291–307, 1970.

[91] B. Khailany, W.J. Dally, U.J. Kapasi, P. Mattson, J. Namkoong, J.D. Owens, B. Towles, A. Chang, and S. Rixner. Imagine : media processing with streams. *Micro, IEEE*, 21(2) :35 –46, mar/apr 2001.

[92] A. A. Khokhar, V. K. Prasanna, M. E. Shaaban, and C.-L. Wang. Heterogeneous computing : Challenges and opportunities. *Computer*, 26(6) :18–27, June 1993.

[93] S. Kirkpatrick. Optimization by simulated annealing : Quantitative studies. *Journal of Statistical Physics*, 34 :975–986, 1984.

[94] O. Klopfenstein. *Optimisation robuste de réseaux de télécommunications*. PhD thesis, Orange Labs , Laboratoire Heudiasyc, UMR CNRS 6599, Université de Technologie de Compiègne, 2008.

[95] B. Korte and J. Vygen. *Combinatorial Optimization : Theory and Algorithms*. Springer, 3rd edition, 2006.

[96] J. Kunkel. COSSAP : A stream driven simulator. In *IEEE International Workshop on Microelectronics in Communications, Interlaken, Switzerland*. IEEE, mar 1991.

[97] E.A. Lee and D.G. Messerschmitt. Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Trans. Computers*, 36(1) :24–35, 1987.

[98] E.A. Lee and D.G. Messerschmitt. Synchronous data flow. *Proceedings of the IEEE*, 75(9) :1235 – 1245, 1987.

[99] E.A. Lee and T.M. Parks. Dataflow process networks. *Proceedings of the IEEE*, 83(5) :773 –801, 1995.

[100] J. Lee, I. Shin, and A. Easwaran. 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, pages 89–98, New York, USA, 2010.

[101] M. Lemerre, V. David, C. Aussagues, and G. Vidal-Naquet. Equivalence between schedule representations : Theory and applications. In *Real-Time and Embedded Technology and Applications Symposium, 2008. RTAS '08. IEEE*, pages 237 –247, 2008.

[102] P. Li, M. Wendt, and G. Wozny. Robust model predictive control under chance constraints. *Computers and Chemical Engineering*, 24(2-7) :829 – 834, 2000.

[103] A. Lisser and F. Rendl. Graph partitioning using linear and semidefinite programming. *Mathematical Programming*, 95 :91–101, 2003.

[104] B. Liu. Fuzzy random chance-constrained programming. *Trans. Fuz Sys.*, 9(5) :713–720, October 2001.

[105] V. M. Lo. Heuristic algorithms for task assignment in distributed systems. *IEEE Trans. Comput.*, 37(11) :1384–1397, 1988.

[106] M. Lombardi, M. Milano, M. Ruggiero, and L. Benini. Stochastic allocation and scheduling for conditional task graphs in multiprocessor systems-on-chip. *Journal of Scheduling*, 13 :315–345, 2010.

[107] D. H. Loughlin and S. Ranjithan. Chance-constrained genetic algorithms. In *GECCO-99 : Proceedings of the Genetic and Evolutionary Computation Conference*, pages 369–376, 1999.

[108] S. Louise. Programmability in the age of the manycore, beyond Stream Programming. *ACM Transactions on Embedded Computing Systems. In print*, 2013.

[109] S. Louise, V. Davidy, and J. Delcoignez. A new paradigm for cache related wcet computation. In *Networks, Parallel and Distributed Processing, and Applications*. ACTA Press, 2002.

[110] J. Luedtke and S. Ahmed. A sample approximation approach for optimization with probabilistic constraints. *SIAM Journal on Optimization*, 19(2) :674–699, 2008.

[111] T. Lundqvist and P. Stenstrom. Timing anomalies in dynamically scheduled microprocessors. In *The 20th Proceedings of IEEE Real-Time Systems Symposium*, pages 12 –21, 1999.

[112] S. Manolache, P. Eles, and Z. Peng. Memory and time-efficient schedulability analysis of task sets with stochastic execution time. *The 24th Euromicro Conference on Real-Time Systems*, 0 :0019, 2001.

[113] S. Manolache, P. Eles, and Z. Peng. Task mapping and priority assignment for soft real-time applications under deadline miss ratio constraints. *ACM Trans. Embed. Comput. Syst.*, 7(2) :19 :1–19 :35, January 2008.

[114] C. Marcon, A. Borin, A. Susin, L. Carro, and F. Wagner. Time and energy efficient mapping of embedded applications onto NoCs. In *ASP-DAC 2005.*, pages 33 – 38 Vol. 1, 2005.

[115] R. Marculescu, U.Y. Ogras, L.-S. Peh, N.E. Jerger, and Y. Hoskote. Outstanding research problems in NoC design : System, microarchitecture, and circuit perspectives. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,*, 28(1) :3–21, 2009.

[116] W. R. Mark, R. S. Glanville, K. Akeley, and M. J. Kilgard. Cg : a system for programming graphics hardware in a C-like language. *ACM Trans. Graph.*, 22(3) :896–907, July 2003.

[117] P. Marwedel, J. Teich, G. Kouveli, I. Bacivarov, L. Thiele, S. Ha, C. Lee, Q. Xu, and L. Huang. Mapping of applications to MPSoCs. In *Proceedings of the seventh IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, CODES+ISSS '11, pages 109–118, New York, NY, USA, 2011. ACM.

[118] A. Mazouz, S. A. A. Touati, and D. Barthou. Study of variations of native program execution times on multi-core architectures. In *CISIS*, pages 919–924, 2010.

[119] A Mehrotra and M.A. Trick. Cliques and clustering : A combinatorial approach. *Operations Research Letters*, 22 :1–12, 1997.

[120] B.L. Miller and H.M. Wagner. Chance constrained programming with joint constraints. *Operations Research*, 13(6) :930–945, 1965.

[121] G.E. Moore. Progress in digital integrated electronics. In *1975 International Electro Devices Meeting,*, volume 21, pages 11 – 13, 1975.

[122] F. Mueller. Timing analysis for instruction caches. *Real-Time Syst.*, 18(2/3) :217–247, May 2000.

[123] S. Murali, L. Benini, and G. De Micheli. A Methodology for mapping multiple use-cases onto Networks on Chips. In *DATE*, pages 118–123. IEEE, 2006.

[124] P. K. Murthy, E.G. Cohen, and S. Rowland. System Canvas : a new design environment for embedded DSP and telecommunication systems.

In *Proceedings of the 9th international symposium on Hardware/software codesign*, CODES '01, pages 54–59, New York, NY, USA, 2001. ACM.

[125] W.A. Najjar, E.A. Lee, and G.R. Gao. Advances in the dataflow computational model. *Parallel Computing*, 25(13) :1907–1929, 1999.

[126] A. Nemirovski and A. Shapiro. Convex approximations of chance constrained programs. *SIAM J. on Optimization*, 17(4) :969–996, December 2006.

[127] H. Orsila, E. Salminen, and T. D. Hämäläinen. Parameterizing simulated annealing for distributing Kahn process networks on multiprocessor SoCs. In *Proceedings of the 11th international conference on System-on-chip*, SOC'09, pages 19–26, 2009.

[128] B. K. Pagnoncelli, S. Ahmed, A. Shapiro, and P. M. Pardalos. Sample average approximation method for chance constrained programming : Theory and applications. *Journal of Optimization theory and Applications*, 142 :399–416, 2009.

[129] B.B. Pal, S. Gupta, and D. Chakraborti. 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)*, pages 1–7, 2010.

[130] M. Pan and C. Chu. FastRoute : A step to integrate global routing into placement. In *IEEE/ACM International Conference on Computer-Aided Design, 2006. ICCAD '06.*, pages 464–471, 2006.

[131] M. Pan and C. Chu. IPR : an integrated placement and routing algorithm. In *Proceedings of the 44th annual Design Automation Conference*, DAC '07, pages 59–62, New York, NY, USA, 2007. ACM.

[132] T. M. Parks. *Bounded Schedule of Process Networks*. PhD thesis, University of California at Berkeley, 1995.

[133] A. Prékopa. The discrete moment problem and linear programming. *Discrete Applied Mathematics*, 27(3) :235–254, 1990.

[134] A. Prékopa. *Stochastic Programming*. Kluwer Acad. Publ., 1995.

[135] J. M. Rabaey, C. Chu, P. Hoang, and M. Potkonjak. Fast prototyping of datapath-intensive architectures. *IEEE Des. Test*, 8(2) :40–51, April 1991.

[136] B. Reistad and D. K. Gifford. Static dependent costs for estimating execution time. *SIGPLAN Lisp Pointers*, VII(3) :65–78, July 1994.

[137] R. T. Rockafellar and S. Uryasev. Optimization of Conditional Value-at-Risk. *Journal of Risk*, 2 :21–41, 2000.

[138] J.A. Roy and I.L. Markov. Seeing the forest and the trees : Steiner wirelength optimization in placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,*, 26(4) :632–644, 2007.

[139] N.V. Sahinidis. Optimization under uncertainty : State-of-the-art and opportunities. *Computers and Chemical Engineering*, 28 :971–983, 2004.

[140] N. Sensen. Lower bounds and exact algorithms for the graph partitioning problem using multicommodity flows. *Lecture Notes in Computer Science*, 2161 :391–403, 2001.

[141] V. Shestak, J. Smith, R. Uml, J. Hale, P. Moranville, A. A. Maciejewski, and H. J. Siegel. Greedy approaches to static stochastic robust resource allocation for periodic sensor driven distributed systems. In *Proceedings of the 2006 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA06)*, pages 3–9, 2006.

[142] A. K. Singh, M. Shafique, A. Kumar, and J. Henkel. Mapping on multi/many-core systems : survey of current and emerging trends. In *Proceedings of the 50th Annual Design Automation Conference*, DAC '13, pages 1 :1–1 :10, New York, NY, USA, 2013. ACM.

[143] O. Sinnen. *Task scheduling for parallel systems.* Wiley-Interscience, 2007.

[144] R. Sirdey. *Contributions à l'optimisation combinatoire pour l'embarqué : des autocommutateurs cellulaires aux microprocesseurs massivement parallèles.* HDR, UTC, France, 2011.

[145] R. Sirdey and P. Aubry. A linear programming approach to general dataflow process network verification and dimensioning. In *Proceedings Third Interaction and Concurrency Experience : Guaranteed Interaction (ICE)*, pages 115–119, 2010.

[146] R. Sirdey and V. David. Approches heuristiques des problèmes de partitionnement, placement et routage de réseaux de processus sur architectures parallèles clusterisées. Technical report, CEA LIST DTSI/SARC/09-470/RS, 2009.

[147] K. Srinivasan and K.S. Chatha. A technique for low energy mapping and routing in Network-on-Chip architectures. In *ISLPED '05*, pages 387 – 392, aug. 2005.

[148] O. Stan, R. Sirdey, J. Carlier, and D. Nace. L'apport de l'optimisation sous incertitudes pour les systèmes temps réel embarqués. In *Ecole de temps réel (ETR)*, 2011.

[149] O. Stan, R. Sirdey, J. Carlier, and D. Nace. A heuristic algorithm for stochastic partitioning of process networks. In *Proceedings of the 16th IEEE International*

Conference on System Theory, Control and Computing (ICSTCC), pages 1–6, 2012.

[150] O. Stan, R. Sirdey, J. Carlier, and D. Nace. The robust binomial approach to chance-constrained optimization problems with application to stochastic partitioning of large process networks. *Submitted to Journal of Heuristics*, 2012.

[151] O. Stan, R. Sirdey, J. Carlier, and D. Nace. A GRASP for placement and routing of dataflow process networks on manycore architectures. In *Eight International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*, 2013.

[152] H.S. Stone. Multiprocessor scheduling with the aid of network flow algorithms. *IEEE Transactions on Software Engineering*, 3(1) :85–93, 1977.

[153] M. W. Tanner and E. B. Beier. 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.

[154] M. Vidyasagar. Randomized algorithms for robust controller synthesis using statistical learning theory. In *Learning, control and hybrid systems*, volume 241 of *Lecture Notes in Control and Information Sciences*, pages 3–24. Springer Berlin / Heidelberg, 1999.

[155] N. Viswanathan and C.C.N. Chu. FastPlace : efficient analytical placement using cell shifting, iterative local refinement and a hybrid net model. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,*, 24(5) :722–733, 2005.

[156] A. Wald and J. Wolfowitz. Confidence limits for continuous distribution functions. *The Annals of Mathematical Statistics*, 10(2) :105–118, 1939.

[157] C. Walshaw, M. Cross, M. G. Everett, S. P. Johnson, and K. McManus. Partitioning & mapping of unstructured meshes to parallel machine topologies. In *Procedeedings on Parallel Algorithms for Irregularly Structured Problems (IRREGULAR)*, pages 121–126, 1995.

[158] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström. The worst-case execution-time problem : overview of methods and survey of tools. *ACM Trans. Embed. Comput. Syst.*, 7(3) :36 :1–36 :53, May 2008.

[159] M. Wipliez and M. Raulet. Classification of dataflow actors with satisfiability and abstract interpretation. *IJERTCS*, 3(1) :49–69, 2012.

[160] H. Xu, C. Caramanis, and S. Mannor. Optimization under probabilistic envelope constraints. *Operations Research*, 60(3) :682–699, 2012.

[161] J. Yang, I. Ahmad, and A. Ghafoor. Estimation of execution times on heterogeneous supercomputer architectures. In *International Conference on Parallel Processing, (ICPP 1993.)*, volume 1, pages 219–226, aug. 1993.