A Heuristic Algorithm for Stochastic Partitioning of Process Networks

Oana Stan, Renaud Sirdey, Jacques Carlier and Dritan Nace

Abstract— **In this paper, we study the problem of partitioning networks of processes under chance constraints. This problem arises in the field of compilation for multi-core processors. The theoretical equivalent for the case we consider is the Node Capacitated Graph Partitioning with uncertainty affecting the weights of the vertices. For solving this problem we propose an approximate algorithm which takes benefit of the available experimental data through a sample-based approach combined with a randomized greedy heuristic, originally developed for the deterministic version. Our computational experiments illustrate the algorithm ability to efficiently obtain robust solutions.**

I. INTRODUCTION

Compilation for embedded systems is an important field of application for operations research techniques, characterized by the need to respect capacity constraints on a set of interdependent resources such as CPU, memory, bandwidth, etc. In the past, optimization methods were mainly applied to compiler backend problems. Nowadays, with the development of complex multi-core architectures, the field of applications for operations research techniques has widened almost to the entire structure of a compiler. The related optimization problems (buffer sizing, task scheduling, placement, routing, etc.) require taking into account a large number of data, which are inherently uncertain and difficult to model with precision.

More precisely, the development of 100+ cores microprocessor architectures [1] has triggered a renewed interest for the so-called dataflow programming models (e.g. [2]) in which one expresses computation-intensive applications as networks of concurrent tasks interacting through (and only through) unidirectional FIFO channels. These models are of the uttermost practical relevance because they relieve the programmer from one of the main pitfalls of parallel programming (having to express explicit complex synchronization schemes). Also, they allow the natural expression of more than enough parallelism for chips in the few hundreds of cores range, at least in the (rather wide) domain of signal and image processing.

A dataflow application can therefore be modeled as a directed graph in which the vertices are the tasks and the arcs are the channels and one of the (numerous) tasks of a dataflow compilation chain consists of mapping this graph onto the hardware resources of the target microprocessor architecture. Thus, in this paper, we study the problem of assigning the weighted vertices of such a graph to a fixed set of partitions, in order 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) [3], [4].

CEA, LIST, Embedded Real Time System Laboratory, Point Courrier 172, Gif-sur-Yvette, 91191 France. oana.stan@cea.fr, renaud.sirdey@cea.fr

UMR CNRS 6599 Heudiasyc, Université de Technologie de Compiègne, BP 20529, 60205 Compiègne, France. jacques.carlier@hds.utc.fr, d.nace@hds.utc.fr

Known, in the deterministic case of single dimensional weights, as the NCGP, this problem has, to the best of our knowledge, received little attention from the stochastic programming community. We present a novel approach for graph partitioning with capacity constraints which is taking into account the uncertainty affecting the node weights. In order to respect as close as possible the real context of our application, a qualitative analysis of the source of uncertainty was performed, leading us to the choice of a non-parametric method. A greedy randomized affinity-based heuristic, described in [5], which proved to be very efficient for the placement of the processes in the deterministic case, was adapted for solving the chance-constrained version of the problem with a preset level of confidence. Therefore, as shown in the sequel, the heuristic algorithm we propose in this paper is providing more robust solutions to the above-mentioned problem.

It should however be emphasized that the non-parametric method we introduce here for solving our chance-constrained problem is general and can be applied, in combination with other approximation algorithms (e.g., metaheuristics), to other optimization problems.

This paper is organized as follows: after a formal definition of our problem in Section 2 and a justification of our work in Section 3, we briefly recall, in Section 4, the notion of relative affinity and the randomized greedy algorithm our approach is using. In Section 5, we discuss the motivations and the basic idea of our method and we introduce the stochastic resolution strategy. Computational experiments are provided and analyzed in Section 6. Finally, concluding remarks and future research directions are presented in Section 7.

II. PROBLEM STATEMENT

The process network partitioning problem can be formally stated as follows.

Let $G = (V, A)$ be a directed graph where the set of vertices V represents the tasks and the arcs $(v, w) \in A$ correspond to the channels of a process network.

Let N be the set of nodes of the parallel architecture on which we want to map our graph.

The resources (essentially memory footprint and computing core occupancy) are given by the set R and the

capacities of the nodes are given by a multi-dimensional array $C \in \mathbb{R}^{+|R|}$. For the sake of simplicity, this study will be limited to the case of homogeneous nodes, hence we suppose all nodes have the same capacity.

Let us also define two functions. $s : V \longrightarrow \mathbb{R}^{+|R|}$, is defined as a size function for the vertices, with $s(v)_r$ being the weight of vertex v for resource r. The second function, defined for the edges, is the affinity function $q : A \longrightarrow \mathbb{R}$ 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 vertices v and w. In the remaining of this paper, we will use the following simplified notation for these functions: $Q_{vw} = q((v,w))$ 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 consider thus consists in finding an assignment of the vertices to the nodes, denoted $f : V \longrightarrow N$, which satisfies the capacity constraints for all resources:

$$\sum_{v \in V : f(v) = n} S_{vr} \leq C_r, \forall n \in N, \forall r \in R, \tag{1}$$

and which minimizes the objective function:

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

III. CONTRIBUTION OF THIS PAPER

In this paper, we present a heuristic algorithm dedicated to the resource-constrained graph partitioning problem which crops up when mapping networks of dataflow processes on a parallel architecture *assuming the resource consumptions of the processes are uncertain*. Our algorithm design methodology consists in leveraging an existing heuristic for the deterministic case without significant destructuring (i.e. at small cost in terms of software engineering) and with acceptable performance hit. Furthermore, this method is applicable to pretty much any existing algorithm. Hence, the contribution of this paper is more centered on demonstrating the practical relevance of our redesign-for-the-stochastic-case methodology than on demonstrating the intrinsic quality of the algorithms involved.

Also, our extension method is non parametric and is therefore suited to application cases where complex multimodal probability distributions occur, a fact which we motivate regarding our own application context.

IV. PRELIMINARIES

A. 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 [5].

Let S and T be two disjoint subsets of V.

Definition 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 2: The total affinity of S (similarly for T) is given by

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

Definition 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.

B. Randomized greedy algorithm for deterministic case

Initially described in [5], the randomized greedy algorithm for processes partitioning in the deterministic case, is based on the *relative affinities* of *admissible assignments* and *admissible fusions*.

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

Definition 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 5: A fusion between the nodes n and m is admissible if for every resource $r \in R$:

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

On empirical ground, the assignments are preferred to 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 6: The vertex v is smaller or lighter than the vertex w if:

$$\max_{r \in R} \frac{S_{vr}}{C_r} < \max_{r \in R} \frac{S_{wr}}{C_r} \tag{2}$$

Definition 7: The node n is more loaded than the node m if for every vertex $v \in V \setminus W$:

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

The algorithm in [5] takes as input the set W, initially equal to V, and the set of nodes N. A basic version of the algorithm consists in:

1) Initialization $W = V$.
2) Assign the first $min(|V|, |N|)$ vertices in lexicographic order to the $|N|$ nodes.
3) Find an admissible assignment (v^*, n^*) ($v^* \in W$, $n^* \in N$), 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^*) ($n_1^* \in N$, $n_2^* \in N$), if any, with maximal relative affinity, for $v \in V \setminus W$,
$$\gamma_2 = \gamma(\{v : f(v) = n_1^*\}, \{v : f(v) = n_2^*\}).$$

5) If $\gamma_1 \geq \gamma_2$ then assign v^* to n^*, else merge n_1^* and n_2^*.
6) If W is empty or there is neither any admissible assignment nor any admissible fusion, stop. Else, go to Step 3.

In order to escape poor quality solutions, a randomized version of the algorithm is executed several times. 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. 2) and several times afterwards using randomly ordered versions of the list of vertices.

This algorithm being given, we can now turn to the stochastic case.

V. CHANCE-CONSTRAINED RANDOMIZED GREEDY APPROACH

With the intention of duly matching the real context of the application, we have performed a qualitative analysis of the sources of uncertainty, mainly the execution times. As described in the next section, the analysis shows the inherent difficulty of obtaining an analytical description of the distributions of the execution times. It also motivates our choice for a model in which the weights of the vertices, dependent on the execution times, are random variables and justifies our recourse to a non parametric sample-based approach.

A. Uncertainty of execution time

One of the main sources of uncertainty in our context lies in the intrinsic indeterminism of the execution times of computing kernels of intermediate granularity (i.e. computing kernels typically requiring a few tens of microseconds). Even if it is reasonable to assume that the probability distributions of execution times have a bounded support (no infinite loops), we have to cope with the fact that these distributions are intrinsically multimodal (due to the presence of data dependent control). Hence, it is difficult to model these probability laws using the usual distributions such as the normal or uniform ones, which are unimodal. Also, in the case of processes networks, we cannot overlook the problem of dependencies between these random variables. Thus, it is appropriate to assume that the execution times are random vectors characterized by complicated multimodal joint distributions.

B. Basic ideas and motivations

Thus, let us now consider the following chance-constrained program:

$$\begin{aligned} \min_{x} \quad & g(x) \\ \text{s.t.} \quad & \mathbb{P}\left(G(x,\xi) \leq 0\right) \geq 1-\varepsilon \end{aligned} \tag{3}$$

In the above program, $x \in \mathbb{R}^n$ is the vector of decision variables, ξ represents a random vector from the probability space $(\Omega, \Sigma, \mathcal{P})$, $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ is the objective function and $G(x,\xi) \in \mathbb{R}^m$ is the constraint function.

$\mathbb{P}(e)$ denotes the probability of the event $e \in \Sigma$ and $0 \leq \varepsilon \leq 1$ is a scalar defining the prescribed probability level.

As one may expect, this class of problem is inherently difficult to address, mainly because of its combinatorial nature. To the best of our knowledge, most of the probabilistic models proposed in the literature (see for instance [6], [7]) assume independence of the components of the random vector or make assumptions about the support of the distribution [8].

On the contrary, the approach we propose is justified by the theory of statistical hypothesis testing and takes into account the important role of experimental data. Additionally, for solving the initial problem (3) no assumptions are being made about the joint distribution of the random variables represented by the vector ξ, in particular with respect to the independence of its components.

C. Statistical hypothesis testing

Let us clarify what it is meant in this paper by the notion of sample and afterwards we will introduce the statistical results on which our method is based. Here, by sample we understand a finite number of realizations or instances of the uncertainty vector ξ.

Given x the decision vector, the random variable χ corresponding to the number of times the inequality $G(x,\xi) \leq 0$ is satisfied on a sample of size NS follows by definition a binomial distribution with parameters NS and p_0. As such, we can search a threshold $k(NS, 1-\varepsilon, \alpha)$ such as, for $p_0 = \mathbb{P}\{G(x,\xi) \leq 0\}$:

$$\mathbb{P}(\chi \geq k | p_0 \geq 1-\varepsilon; NS) \leq \alpha$$

where $\alpha \in]0,1[$ is a small probability of error (for example 0.05 or 0.01). The threshold k should be set sufficiently high so that if we observe a number of constraint violations which is superior to k, the probability that this number is generated by a binomial distribution $\mathcal{B}(NS, 1-\varepsilon)$ is sufficiently low. Hence, we can conclude, with a *confidence level* of at least $1-\alpha$, that $p_0 \geq 1-\varepsilon$.

We can also establish in advance the minimal sample size necessary for a prescribed level of the probability $\varepsilon \in]0,1[$ and a required confidence level $\alpha \in]0,1[$. In particular, if:

$$\mathbb{P}(\chi = NS - 1 | p_0 \geq 1-\varepsilon; NS) \geq \alpha$$

then we can affirm that the sampling size is insufficient (which is true for $NS = 10$ and $NS = 20$). The above formula provides an easy way to determine the minimal number of realizations that need to be drawn in order to achieve the desired confidence level, established based only on the parameters ε and α.

D. Randomized greedy algorithm: stochastic case

The statistical hypothesis test explained previously can be easily integrated in a heuristic approach by counting the number of constraint violations. For the stochastic version of our graph partitioning problem, formally stated in Section II, 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 $k = 1$ to NS, let $\tilde{S}_{vr}^{(k)}$ be the realization of the k-th observation.

Also denote 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 (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 the given confidence level, one can simply, at every step of the algorithm, redefine the notions of *admissible assignment* and *admissible fusion*.

Definition 8: An assignment of vertex v to node n is stochastically admissible if :

$$\sum_{k=1}^{NS} \chi(A \vee B) \leq NS - k(NS, 1-\varepsilon, \alpha)$$

with

$$A = \{\exists n' \neq n, \exists r : \sum_{w: f(w)=n'} \tilde{S}_{wr}^{(k)} > C_r\}$$

and

$$B = \{\exists r : \tilde{S}_{vr}^{(k)} + \sum_{w: f(w)=n} \tilde{S}_{wr}^{(k)} > C_r\}$$

and where $\chi(A \vee B) = 1$ if and only if the predicate $\{A \vee B\}$ is true.

This definition could be further simplified by the use of a boolean matrix of size $|N| \times NS$ indicating for the partial current partitioning if, for every node, the sample k has already induced a violation. With the use of this array, the computation of an admissible assignment increases in complexity linearly, with a factor of NS, compared to the deterministic case.

Definition 9: A fusion between nodes n and m is stochastically admissible if:

$$\sum_{k=1}^{NS} \chi(A \vee B) \leq NS - k(NS, 1-\varepsilon, \alpha)$$

with

$$A = \{\exists n', r : \sum_{w: f(w)=n'} \tilde{S}_{wr}^{(k)} > C_r\}$$

and

$$B = \{\exists r : \sum_{w: f(w)=n} \tilde{S}_{wr}^{(k)} + \sum_{v: f(v)=m} \tilde{S}_{vr}^{(k)} > C_r\}$$

and where $\chi(A \vee B) = 1$ if and only if the predicate $P_f = \{A \vee B\}$ is true.

Analogously, we can simplify P_f by using the same boolean matrix $|N| \times NS$.

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

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.

Definition 10: The vertex v is smaller or lighter in average than the vertex w if:

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

Definition 11: The node n is more loaded in average than the node m if, for every vertex $v \in V \setminus W$:

$$\max_{r \in R}(1 - \frac{\sum_{f(v)=n} \tilde{S}_{vr}^{(k)}}{NS * C_r}) < \max_{r \in R}(1 - \frac{\sum_{f(v)=m} \tilde{S}_{vr}^{(k)}}{NS * C_r})$$

The above definitions are then easily integrated in the algorithm, described in Section IV-B, without any major modifications. By using the statistical hypothesis testing within a heuristic approach, we also overcome the computational effort of taking into account the uncertainties of the weights of the vertices. Additionally, we could further improve the performances of the heuristic by parallelizing the computations of admissible assignments and of admissible fusions.

VI. COMPUTATIONAL RESULTS

In this section, we report on the computation experiments of the extension of the randomized greedy algorithm for the stochastic case of 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.

Benchmark and Uncertain Parameters Generation

Since, to the best of our knowledge, there are no stochastic instances defined for the graph partitioning problem, 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, which are representative in size of 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 composed by instances publicly available (between 124 and 1000 vertices), defined in Johnson et al. [9], 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 also be noted that the Grid 23×23, with 529 vertices and 16 nodes, is the closest in size to the real instances in our application context and that for both sets, we consider the case of mono-dimensional resources.

The random variables representing the weights of the vertices are generated 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 $[1.1, 1.2]^{|V|}$.

Results for the deterministic version

Table I shows the experimental results obtained by applying the randomized greedy heuristic for minimizing the internodes bandwidth in the deterministic case on some simple examples of grids. All the results were computed for the monodimensional case (the capacity of each node is indicated in column "C") with unitary weights edges and vertices, the column "Multi" in Table I showing the solutions found by running the multi-start version of the heuristic (with 10 iterations) and the column "Time" showing the running time for one iteration in average over 10 iterations. The same multi-start version of the heuristic for the monodimensional deterministic case was applied on the larger sizes instances of Johnson et al. [9], the solutions values found having an average differential approximation ratio [10] of 5.22% compared to the best known value.

TABLE I

COMPUTATIONAL RESULTS OF THE ALGORITHM IN THE DETERMINISTIC CASE

Inst.	#Vertices	#Nodes	C	Multi	Time
Grid 4×4	16	4	4	8	≈ 0
Grid 10×10	100	5	20	28	≈ 0
Grid 23×23	529	14	40	150	0.12 s

These results are only provided for self-containedness purpose and for serving, in the next section, as a possible measure of the price of robustness of the solutions obtained by our algorithm which is taking into account the uncertainties affecting the weights of the vertices.

Results for the stochastic version

We have tested our adaptation of the algorithm for the stochastic case on the same benchmark varying the parameters ε and α in the range $\{0.01, 0.05\}$. To use the benchmark as a set of stochastic instances, we have considered that the weights of the vertices are random variables with the aforementioned bimodal distribution and we have generated corresponding samples of size 100 and respectively 1000.

The method has been implemented in C and, for each instance, 10 random iterations of our algorithm were executed.

Tables II - IV summarize the numerical results for the grid problems for different values of the parameters NS, ε and α. As already reported, extensive computational experiments were also performed on the benchmark instances of Johnson [9].

For each instance from the data sets, we performed two tests. The first one consists in keeping the same node capacity as for deterministic case (see, for example, column C in Table I) and increasing the number of nodes until the probabilistic constraint is satisfied. The numerical results of this test, reported in section "1st test" of Tables II - IV are: the minimal number of nodes for which the probabilistic constraint is respected (column "#nodes"), the value of the solution (column "sol") and the execution time for one iteration in average over 10 iterations (column "time"). For the second test, we keep the same number of nodes as in the deterministic case, but we increase the capacity of all nodes until finding a feasible solution, satisfying the probabilistic constraint. The results of this second test, reported in section "2nd test" of Tables II - IV are: the minimal capacity of each node for which we obtain a feasible solution (column "C"), the solution value (column "sol") and the execution time for one iteration in average over 10 (column "time").

TABLE II

COMPUTATIONAL RESULTS OF STOCHASTIC METHOD FOR $NS = 100$, $\varepsilon = 0.05$, $\alpha = 0.05$

	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

TABLE III

COMPUTATIONAL RESULTS OF STOCHASTIC METHOD FOR $NS = 1000$, $\varepsilon = 0.05$, $\alpha = 0.05$

	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 IV

COMPUTATIONAL RESULTS OF STOCHASTIC METHOD FOR $NS = 1000$, $\varepsilon = 0.01$, $\alpha = 0.01$

	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

Before continuing with the analysis of our results, it is worthwhile noting that the solutions obtained in the second experiment, 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 III for the Grid 10×10, for finding a feasible solution, we must add one more node but in this case the solution found is too conservative since all the constraints are verified. Of course, neither increasing the number of nodes nor increasing the node capacity make sense in practice. Our intent with these experiments is to get an idea of the cost of the robustness independently of concrete systems constraints.

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

As expected, for the first test, the quality of the stochastic solutions is less than that for the deterministic version as well as for the solutions found by the second test. Instead, the stochastic solutions of the second test are close in quality with the solutions found in the deterministic case. By analyzing the 25 instances of Johnson, we observe that there are 13 and respectively 15 instances with a gap in

the 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, we observe slight increase in the number of solutions close to the deterministic solutions. These results suggest that even if, in some cases, the quality of the stochastic solutions can be less satisfying than the quality of the deterministic solutions, the solutions remain of a good quality and, more importantly, they are guaranteed with a target probability level. Therefore our stochastic algorithm is more suitable to address applications with uncertain data.

By comparing the quality of solutions for different values of the input parameters (NS, ε, α) it comes out for both benchmarks 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.

Concerning the computation time, we have observed an average execution time of 48.04 sec. for a sample of size 1000 against 25.93 sec. for a sample of size 100 for the Johnson instances in the second experiment ($\varepsilon = 0.05$, $\alpha = 0.05$). 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, even at the beginning of the development cycle. The running times found for the stochastic version of the algorithm confirm the theoretical remarks (see Section V-D) on a linear increase in complexity with a factor of NS in comparison of the deterministic case.

In our first experiment, we were interested in the number of nodes needed for the stochastic case compared to the deterministic one. Our computational tests show that the ratio between the number of nodes for stochastic partitioning and the number of nodes for deterministic partitioning of 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 II - IV 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 ≈ 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).

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.

VII. CONCLUSION

In this paper, we addressed the problem of chance-constrained partitioning networks of communicating processes, which arises in dataflow compilation for embedded parallel systems. After a brief analysis of the sources of uncertainties, we proposed a heuristic algorithm, combining statistical hypothesis testing with a randomized greedy method originally designed for the deterministic case. In our experiments the solutions computed have a quality comparable to those computed by the deterministic version (a property which cannot be expected to hold in general for optimization taking into account data uncertainties), and moreover they are statistically guaranteed at confidence level $1-\alpha$. Furthermore, this approach can solve, with an acceptable solution quality, confidence level and computation time, problems representative in size of our application context.

Additionally, this approach to chance-constrained programming is general and could be applied to leverage algorithms for the deterministic case for solving the stochastic version of other combinatorial problems (when obtaining experimental data is not an issue).

In further work, we plan to investigate two directions. The first one is to design a parallelized and more efficient implementation of our method. The other direction concerns a more thorough analysis and characterization of execution time distribution, using methods of static code analysis. This will allow us to have a more accurate characterization of the uncertainties affecting the weights of the processes.

REFERENCES

[1] Kalray, "Introduction to the MPPA technology. A technical overview," Tech. Rep. KETD-58, Kalray S. A., 2011.

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

[3] M. Garey, D. Johnson, and L. Stockmeyer, "Some simplified NP-complete graph problems," *Theoretical Computer Science*, vol. 1, no. 3, pp. 237–267, 1976.

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

[5] R. Sirdey and V. David, "Approches heuristiques des problèmes de partitionnement, placement et routage de réseaux de processus sur architectures parallèles clusterisées," tech. rep., CEA LIST DTSI/SARC/09-470/RS, 2009.

[6] D. Bertsimas and M. Sim, "The price of robustness," *Operations Research*, vol. 52, no. 1, pp. 35–53, 2004.

[7] A. Ben-Tal and A. Nemirovski, "Robust solutions of linear programming problems contaminated with uncertain data," *Mathematical Programming*, vol. 88, pp. 411–424, 2000. 10.1007/PL00011380.

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

[9] E. Johnson, A. Mehrotra, and G. L. Nemhauser, "Min-cut clustering," *Mathematical Programming*, vol. 62, pp. 133–151, October 1993.

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