4OR-Q J Oper Res
DOI 10.1007/s10288-015-0280-0

INDUSTRY

A GRASP metaheuristic for the robust mapping and routing of dataflow process networks on manycore architectures

Oana Stan · Renaud Sirdey · Jacques Carlier · Dritan Nace

Received: 3 February 2014 / Revised: 29 August 2014

Abstract In this paper, we study the problem of joint placement and routing, both in the deterministic and stochastic cases, arising in the field of compilation of dataflow applications for manycore architectures. A GRASP algorithm is first proposed for solving the deterministic version and extended afterwards to treat the chance-constrained program with uncertainty affecting the weights of a dataflow process network. Extensive computational results, on representative synthetic benchmark and real data, illustrate the practical relevance of the approach, as well as the robustness of the obtained stochastic solutions.

Keywords Placement and routing · Chance-constrained programs · Manycore · Robust optimization

Mathematics Subject Classification 90C15 · 90C59 · 68N20

1 Introduction

This article addresses the problem of application mapping, classified as "one of the most urgent problems to be solved for implementing embedded systems" (Marwedel et al. 2011; Marculescu et al. 2009). There are different mapping methodologies varying in function of application and architecture models, constraints and assumptions

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

J. Carlier · D. Nace
UMR CNRS 6599 Heudiasyc, Université de Technologie de Compiègne,
BP 20529, 60205 Compiègne, France

Published online: 28 January 2015

imposed by the system, information available about the platform, etc. However, the routing aspect has almost always been neglected and the inherent uncertainty affecting the problem data (such as the execution times of the tasks) has been ignored.

The purpose of this study is to propose a *routable placement* method on a clusterized manycore architecture of applications expressed in ΣC (Aubry et al. 2013; Goubier et al. 2011), a particular type of Dataflow Process Networks (DPNs). As such, it provides an alternative approach to the sequential placement and routing steps of the resource allocation compilation step of a ΣC program, which targets specific application domains (e.g. multimedia and networking) characterized by high bandwidth demands.

Even if the two NP-hard 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 infrastructure (here a Network-On-Chip) topology, both for the *deterministic* and *stochastic* cases. Therefore, the method we propose remains generic and can be applied to applications expressed as DPN on any kind of architecture model.

It is also worth mentioning that the current approach is extensible to solve the mapping of applications expressed using other models of dataflow networks inclusive Kahn processes. In fact, as we will see further on, the placement and routing process we treat here occurs in the compilation flow after the analysis of the properties of the dataflow model. Consequently, the specific properties associated with a particular dataflow model have been exploited before and served to generate the input data for our problem.

A GRASP heuristic has been conceived for solving the problem in the deterministic case and afterwards adapted to solve the stochastic version, when the weights of the tasks are supposed to be uncertain parameters.

The rest of this paper is organized as follows. The next section is useful in situating the application context: compilation of DPNs for manycore systems. Section 3 gives a formal description of the problem while the following one describes similar existing approaches and provides a brief overview of the techniques for optimizing under uncertainty. In Sects. 5 and 6, we present in details the structure of our GRASP method for respectively deterministic and stochastic cases. Section 7 provides the results of computational experiments conducted on synthetic benchmarks as well as on a real relatively complex ΣC application. Some final remarks and future research perspectives are presented in the last section.

2 Context and research motivations

Nowadays, the embedded systems industry is revolutionized by the emergence of massively multi-core (manycore) architectures. Designed as a solution to reduce the Moore's gap,[1] they are composed of at least a dozen of parallel processing elements

[1] Despite the exponential growth of the number of transistors which can be placed on an integrated circuit (according to Moore's law), the performance of practical computing systems does not follow the same growth rate.

(cores), a mix of local and shared memory, distributed global memory or multilevel cache hierarchy and an infrastructure for inter-cores communication such a Network-On-Chip (NoC).

The Kalray's MPPA-256 (Dupont de Dinechin et al. 2013), the platform used in this paper for the real user case, is one of the first homogeneous embedded manycore, released in 2013 and manufactured using 28 nm CMOS technology. This single-chip manycore processor is organized as 16(4×4) computing clusters connected through a bidirectional NoC with a 2D torus topology. As the basic processing unit of the MPPA chip, each computing cluster integrates as main components 16 processing engines (PE) (also called cores), one resource management (RM) core, a shared memory and a direct memory access (DMA) engine for transferring data.

Programming 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 dataflow paradigm seems to be a good candidate for programming manycore applications as it satisfies most of the properties stated before. With the first models emerging in the early 1970s, dataflow languages (Lee and Parks 1995) provide an efficient and simple solution to express programs, which can be executed on a parallel architecture, without worrying about data synchronization.

Traditional programming imperative languages (like C or Java) even with SMP extensions are based on a sequential von Neumann architecture and therefore are of lower productivity for writing effective parallel programs for manycore systems.

Moreover, the Message Passing libraries such as MPI associated with the traditional languages and used currently on distributed systems require explicitly managing communications and synchronizations between tasks.

An example of a recent dataflow model and language is ΣC [e.g. Goubier et al. (2011)], a language and model which allows to perform a formal analysis for verifying properties like absence of deadlock or memory bounded execution.

Using a dataflow model, an application is described as a static instantiation graph of concurrent tasks (agents) interacting through unidirectional FIFO channels. Once the application has been designed and implemented using a dataflow programming language, it is the role of the compilation chain to make the connection with the specific execution model for the embedded manycore target.

Let us exemplify the compilation process through ΣC compilation chain, organized into four passes:

- *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 either for off-line execution (the instantiation codes of the agents), or for further refinement.
- *Compilation of the parallelism.* The purpose of the second pass, the ΣC middle-end, is to instantiate and connect the agents, by executing at compile time the corresponding codes generated by the first pass.

Once the construction of the application graph is complete, parallelism reduction techniques by pattern matching (Carpov et al. 2013) are applied and a safe computation of a deadlock-free lowest bound for the buffers sizes of the links is also performed.

– *Resource allocation.* The third pass is in charge of resource allocation (in the larger sense). First, it supports a dimensioning of communication buffers taking into account the execution times of the tasks and the application requirements in terms of bandwidths (non functional constraints). Next, in order to realize a connection with the execution model, it constructs a folded unbounded partial ordering of task occurrences (and thus, finitely representable).
  This pass is also responsible of placement and routing, with the objectives of grouping together (under capacity constraints for each cluster of the architecture) the tasks which communicate the most, mapping these groups of tasks to the clusters and, finally, computing routing paths for the data traversing the NoC.
– *Runtime generation and link edition.* The last pass, the ΣC back-end, is responsible of generating the final C code and the runtime tables. Also, during this stage and using C back-end compiler tools, link edition and loadbuild are realized.

The optimization problem we consider here, related to the third pass of compilation, consists in the joint placement and routing of Dataflow Process Networks (DPN) on a homogeneous clusterized manycore architecture in which the cores are organized in clusters communicating through an asynchronous Network-On-Chip.

Let us now formally describe the problem.

3 Problem statement

The static mapping of tasks from a DPN onto the network of clusters consists in a tasks placement such as the total bandwidth used by the application is minimal and for each pair of communicating tasks, there exits a feasible routing path between tasks situated on different clusters.

The clusterized architecture is 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 $C_n \in \mathbb{R}^{+|R|}$.

As our initial target architecture is made of homogeneous clusters linked by a homogeneous NoC, the remaining of this paper will be restrained 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.

Additionally, the extension of our method to heterogeneous architectures is not difficult, one mainly having to modify the admissibility tests (i.e. take into account different capacity constraints for each cluster and different maximal bandwidths for each arc). As such, the generalization of our approach to non homogeneous case do not change the problem structure or its size.

Let $DPN = (V, E, S, Q)$ be a directed graph representing 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.

Since the core occupancies of the tasks are computed based on their execution times, one of the main sources of uncertainty for combinatorial problems arising in embedded field, we consider the stochastic case with variations on the weight of tasks, as detailed in Sect. 6.2.

$Q : E \longrightarrow \mathbb{R}$ characterizes the communication 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 $g : V \rightarrow N$ be a mapping of tasks to the nodes. As such, we are interested in finding an admissible assignment g of tasks to nodes minimizing the sum of inter-tasks communications:

$$\sum_{(tt')\in E : g(t)\neq g(t')} q_{tt'}. \tag{1}$$

Since the cost of tasks communications situated on same cluster is several order of magnitude smaller than the cost of communications between tasks assigned on different clusters (memory-based versus NoC-based communication), the former is ignored in the overall objective function.

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

$$\sum_{t\in V : g(t)=n} s_{tr} \leq C_{nr}, \forall n \in N, \forall r \in R, \tag{2}$$

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

$$\{\forall (t, t') \in E \text{ and } g(t) \neq g(t') \text{ and } q_{tt'} \geq 0\} : \exists \text{route(t,t')} \tag{3}$$

which respects the maximal capacity B_a of the links of the network.

As such, the last condition verifies if all the communications 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.

The search of routes is limited to unsplittable flow for simplifying the communication protocols and also, for avoiding the problems of restoring the order of arrival of data packets. If, instead, multi-flow is considered for the routing, a software mechanism is necessary in order to handle the order of arrival and to rearrange the packets.

Since the tasks mapping is equivalent to the Node Capacitated Graph Partitioning problem which is NP-hard (Garey et al. 1976) and unsplittable flow problem can be restricted to the Directed Edge Disjoint Paths problem, also NP-hard (Korte and Vygen 2006), the joint problem is straightforwardly NP-hard in the strong sense.

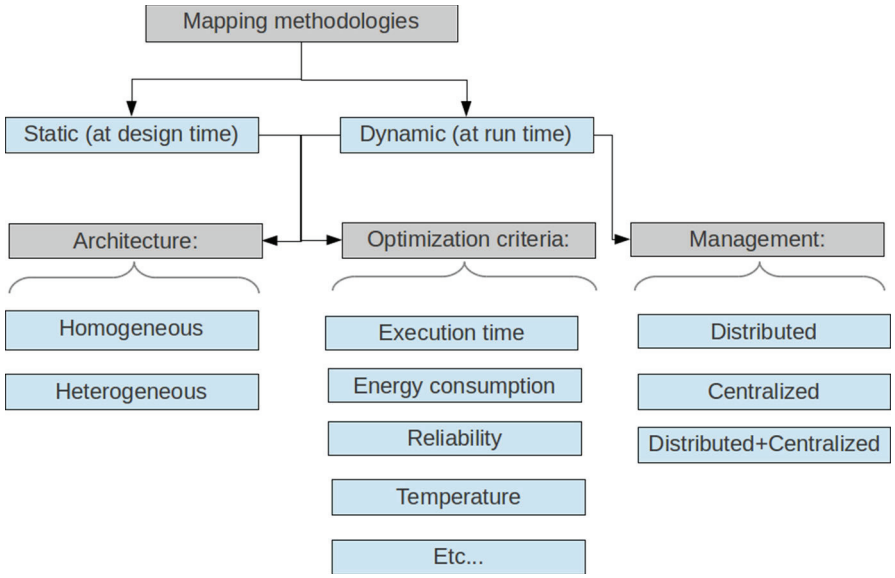

Fig. 1 Classifying mapping approaches [extended version of Singh et al. (2013)]

Regarding the size of instances specific to our context of application, our method has to be able to map networks of processes with a few hundreds of tasks on architectures having at least a dozen of nodes. An example of a real application a compilation chain has to treat, which will be used across this study for an experimental validation, is a motion target dataflow which has to be placed on a NoC with a bi-dimensional 4×4 torus topology.

Let us now present some existing work on mapping methodologies.

4 Related work

4.1 Mapping methodologies

As shown in Fig. 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 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 multicore systems and can be optimized for different optimization metrics.

Since we are interested in static mapping, the following section presents some of the methods belonging to this category for both deterministic and stochastic case. For a detailed survey on mapping strategies, please refer to Singh et al. (2013).

4.1.1 Static mapping approaches for the deterministic problem

Actually, even if most of the existing studies treating task mapping belong to this category, they remain different in the architectures they target (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 (1977) and Lo (1988)], the routing aspect has been often neglected, the scheduling problem drawing more the researchers attention.

For the mapping of applications expressed as dataflow process networks and for which the target architecture is a multicore system, we can cite Orsila et al. (2009), Castrillon et al. (2012), Bonfietti et al. (2010), Choi et al. (2012), Galea and Sirdey (2012). In Orsila et al. (2009), a simulated annealing algorithm is proposed for distributing Kahn Process Networks (the most general dataflow model) on Multiprocessors SoCs (MpSoCS) with at most four Processing Elements (PE) connected with dual shared bus. Galea and Sirdey (2012) proposes a parallel simulated annealing approach for the DPNs mapping on a square torus architecture. Since this method is quite computationally demanding (roughly 20 min 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. Castrillon et al. (2012) 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 Bonfietti et al. (2010), the authors address both mapping and scheduling of applications expressed as SDF (Synchronous DataFlow, models in which a fixed amount of data is consumed and produced at each firing of a task) 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 (Choi et al. 2012), 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.

Several approaches (Marcon et al. 2005; Srinivasan and Chatha 2005; Murali et al. 2006; Hu and Marculescu 2005) 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.

Instead, we consider that the manycore specification and in particular NoC characteristics such maximal bandwidth for links are rigid. As such, the placement and the routing of tasks are realized afterwards (without worrying about scheduling) during the compilation process of a dataflow application.

The only similar approach, of which we are aware of, treating the same problem under the same constraints as ours is Sirdey (2011) 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 disrupt the structure of 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 we cannot route because the flows between the nodes of the network exceed the maximal bandwidth capacity for the links B_a.

Also, the GRASP algorithm for the deterministic placement and routing was first introduced in Stan et al. (2013). For the current paper, we have enriched the post-optimization with the addition of an option for 1-OPT search and we have also performed extensive tests on a new and a larger benchmark, composed of random generated instances. Moreover, we considered here the stochastic aspect of the problem when the uncertainty is on the weights of the tasks, which was, at our knowledge, never been treated before.

4.1.2 Static mapping approaches for the stochastic problem

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 Manolache et al. (2008), stochastic mapping and priority assignment of graph tasks on a multiprocessor hardware architecture is performed via a tabu search heuristic with the goal of optimizing the ratio of deadlines missed. The underneath assumption is that for each task and each processor, a set of probability density functions for the execution time is available.

Lombardi et al. (2010) 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 analysing 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.

Shestak et al. (2006) 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. The obtained approximation of the cumulative density function is further employed by the four greedy heuristics the authors design.

Therefore, we can 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. Afterwards, we will present its adaptation to the stochastic case, for which we used the robust binomial approach, a optimization method we developed, first described in Stan et al. (2012).

5 GRASP for the deterministic joint mapping and routing

5.1 Preliminaries

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, monoroute the flows on the Network-On-Chip. 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 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 the 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 well suited choice to tackle this problem especially for the beginning of the development cycle of an application.

Introduced in the nineties by Feo and Resende (1995), GRASP (Greedy Randomized Adaptive 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 neighbourhood of the current solution is investigated in the search of better solutions. At the end, the best overall solution is kept as the result.

From a software engineering point of view, a GRASP is made of simple algorithmic components which often already exists and which then can be leveraged almost directly in this metaheuristic framework (and, as shown in this paper, further generalized to cope with uncertainty). For our problem, we inspired our construction phase from the already existing greedy algorithm (Sirdey 2011), which was used for the partitioning of dataflow networks in the deterministic case. Also, due to its speed of execution, we considered it more adapted to be applied at the beginning of the compilation cycle, especially if we want to consider the stochastic aspect of the parameters, known to

increase the complexity of a problem. For the end of the compilation cycle, when the computing times are not a concern, other approximation methods [like, for example, the simulated annealing from Galea and Sirdey (2012)] could be extended to guarantee the routing in the deterministic case and afterwards, adapted for the stochastic case.

Algorithm 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 the 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 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 notion of affinity, initially introduced in Stan et al. (2012).

Let S and T be two disjoint subsets of V and $\delta(S, T) = \{(v, w) : v \in S; w \in T\}$.

Definition 1 The relative affinity of S for T is defined as

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

where $\alpha(S, T)$ is the total affinity equal to $\sum_{(v,w)\in\delta(S,T)} q_{vw}$.

5.2 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 (Stan et al. 2012). 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 $\{n_1, n_2, \ldots, n_m\} \in |N| \times |N| \times \cdots |N|$ with $m \in \{0, |N| - 1\}$, $n_1 = g(s(f))$ and $n_m = g(d(f))$, such that $\forall i = \{1, \ldots, m-1\}$, $C_{r_{(n_i, n_{i+1})}} \geq w(f)$ 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 the respect of capacity resources for the clusters and also the existence of a routing.

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

Definition 2 An assignment of task t to node n is admissible if it satisfies the capacity constraints for node n:

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

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 3 A fusion between the nodes n and m is admissible if:

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

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

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.

The overall framework of the greedy randomized construction algorithm is presented in Algorithm 2. Initially, a partial solution is set as the first $\min(|V|, |N|)$ tasks in lexicographic order on the decreasing tasks weights assigned to the N nodes with the condition that this initial mapping is also routable.

As such, by placing the heaviest tasks first (one on every node), we assure a first admissible routable assignment whose eventually poor quality could be later improved by a fusion between two nodes. This initial assignment is intended to cope with the bin-packing side of the problem in the case when a few large tasks are present.

Then, the list $[rcl]$ of the 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.

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 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.2) and admissible fusions (cf. Def.3)
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: $W = W \backslash \{t_i\}$, 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.

After each assignment or fusion, G' and F are updated accordingly, by modifying C_{r_a} and by adding and/or removing flows.

To take advantage of the randomization applied in step 4, the overall GRASP is executed several times (a maximal number of iterations) and the best solution is kept at the end. If, not even after this multi-execution, an admissible assignment is not found, i.e. W is empty, the application can not be automatically mapped on the target architecture and a compilation error occurs.

5.3 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 neighbourhood 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 neighbourhoods 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 (Feo and Resende 1995).

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 Sect. 5.1).

Once the set EX_t is constructed, only *admissible transfers* or *admissible exchanges* are analysed. 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 For a given assignment g, a transfer of task t to a node n is admissible if:

- the capacity of node n remains satisfied 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: $S' = S + \sum_{g(t')=g(t)}(q_{tt'} + q_{t't}) - \sum_{g(t')=n}(q_{tt'} + q_{t't})$.

Definition 5 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 satisfied

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

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

- 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 new value of the solution when moving t to $g(t')$ and t' to $g(t)$ will be: $S' = S + \sum_{g(t)=g(t_i)}(q_{tt_i} + q_{t_i t} - q_{t't_i} - q_{t_i t'}) + \sum_{g(t')=g(t_i)}(q_{t't_i} + q_{t_i t'} - q_{tt_i} - q_{t_i t})$.

6 Extended GRASP for the stochastic joint mapping and routing

As already mentioned, one of the main sources of uncertainty for optimization problems related to compilation of dataflow applications for manycore lies in the intrinsic indeterminism of execution times for computing kernels of intermediate granularity. This indeterminism is due in part to some of the characteristics of the processor architecture such as the cache memories and memory access controllers and is also inherently due to data dependent control flows (conditional branches and loops).

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 the distributions are intrinsically multimodal, multidimensional and difficult to fully characterize. An easy example, showing the dependency issue for these random variables, consists in a 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. However, in the context of an iterative compilation, we dispose of observations of these execution times, when performing tests on the target architecture and of which we could eventually take advantage for the joint mapping and routing problem.

The execution times taking part in the computation of computing core occupancy of the tasks, we consider the stochastic case in which the weights of tasks s_{tr} are random variables. Therefore, we obtain the associated chance-constrained problem, in which constraints (2) are being replaced by the probability constraints for the capacities of the clusters:

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

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

In order to solve the chance-constrained version of the mapping and routing problem, while respecting the characteristics of the uncertain variables, we applied the robust binomial approach, introduced in Stan et al. (2012) and briefly described in the next section.

6.1 Preliminaries: the robust binomial approach

The type of stochastic problem we deal with here is a chance-constrained program (Prékopa 1995), with the following general form:

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

where $x \in \mathbb{R}^n$ is the decision variable vector, $\xi \in \Omega \longrightarrow \mathbb{R}^D$ represents a random vector and $g : \mathbb{R}^n \longrightarrow \mathbb{R}$ is the objective function.

We suppose that there exists a probability space $(\Omega, \Sigma, \mathbb{P})$, with Ω, the sample space, Σ, the set of events, i.e. subsets of Ω, and $\mathbb{P}$, the probability distribution on Σ where $\mathbb{P}(e)$ of an event e is the probability measure on the set Σ.

Also, in the following, we denote by $\tilde{z}$ the realization of a random variable z.

Most of the existing studies for solving chance-constrained programs 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 (as is the case for the execution times).

An extension of the scenario method (Calafiore and Campi 2006), justified by the theory of statistical hypothesis testing, the *robust binomial approach*, takes into account the role of experimental data, without any assumption on the joint distribution of the random vector ξ (in particular on the interdependence of the components of ξ).

The only major assumption made is the existence of a representative sample for ξ, of sufficiently large size, composed of independent and identically distributed (i.i.d.) observations: $\xi^{(1)}, \ldots, \xi^{(NS)}$.

Given x the decision vector, the random variable χ corresponding to the number of times the inequality $G(x, \xi) \leq 0$ is satisfied by the sample follows a Binomial distribution with parameters NS and p_0 ($\chi \sim \mathcal{B}(NS, p_0)$). We can choose a threshold $k(NS, 1-\varepsilon, \alpha)$ (for simplicity sake, we will refer to it as k) so that the probability we accept the constraint by error is smaller than a fixed α, in which case p_0 is smaller than $1-\varepsilon$. The parameter α, to which we refer as the confidence parameter, can be interpreted as the type I error of a statistical hypothesis test.

We can obtain thus an approximation to the initial chance-constrained program, in which the number of realizations $\tilde{\chi}$ respecting the capacity constraints is compared with the parameter k, fixed in advance in function of the values of NS, ε and α. If a feasible solution is found for this program, then we can affirm that we found a solution which is also feasible for the initial stochastic program with a minimal probability of $1-\varepsilon$ and a level of confidence of $1-\alpha$.

Due to the induced complexity (depending on the size of the sample), the robust binomial approach is better suited for approximation algorithms. For an existing (meta)heuristic conceived for solving a given deterministic problem , it is relatively easy to adapt it using the robust binomial approach for solving the stochastic version of the same problem. In the case of a constructive algorithm, there is the need to modify the way feasibility of solutions is decided, by changing the oracle deciding the admissibility of a solution. As for the heuristics based on neighbourhoods, the robust binomial approach must be integrated when establishing the list of potential neighbours.

Let's go back now to our problem of joint mapping and routing and the necessary changes in our GRASP algorithm, when applying the robust binomial approach.

6.2 Changes in the initial algorithm

We 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 $\tilde{s}_{rt}^{(1)}, \ldots, \tilde{s}_{rt}^{(NS)}$.

As such, we can use the robust binomial approach and obtain robust solutions, guaranteed with a probability level of $1-\varepsilon$ and a confidence level of $1-\alpha$, 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.

Definition 6 An assignment of task t to node n is stochastically admissible if:

- the sum

$$\sum_{i=1}^{NS} \tilde{\chi}\left(\left\{\forall n' \neq n, \forall r : \sum_{t':g(t')=n'} \tilde{s}_{t'r}^{(i)} < C_r\right\} \wedge \left\{\forall r : \tilde{s}_{tr}^{(i)} + \sum_{t':g(t')=n} \tilde{s}_{t'r}^{(i)} < C_r\right\}\right)$$

  is superior or equal to $k(NS, 1-\varepsilon, \alpha)$, where $\tilde{\chi}(\mathcal{P}_a) = 1$ 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 $t' \in V \backslash W$ with $g(t) \neq g(t')$ and $(tt') \in E$:

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

Definition 7 A fusion between the nodes n and m is stochastically admissible if:

- the sum

$$\sum_{i=1}^{NS} \tilde{\chi}\left(\left\{\forall n', r : \sum_{\substack{t:g(t)=n' \\ n' \neq n,\ n' \neq m}} \tilde{s}_{tr}^{(i)} < C_r\right\} \wedge \left\{\forall r : \sum_{t:g(t)=n} \tilde{s}_{tr}^{(i)} + \sum_{t':g(t')=m} \tilde{s}_{t'r}^{(i)} < C_r\right\}\right)$$

  is superior or equal to $k(NS, 1-\varepsilon, \alpha)$, where $\tilde{\chi}(\mathcal{P}_f) = 1$ 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 in Algorithm 2 are during the steps 2 and 3 in which the admissibility criterion is used.

As for the post-optimization step, we used a simple neighbourhood in which a new solution is built from the previous by exchanges of pairs of tasks or transfers of a single task to another cluster. This 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 in order to apply the robust binomial approach.

Definition 8 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 $\sum_{i=1}^{NS} \tilde{\chi}(\mathcal{P}_a) \geq k(NS, 1-\varepsilon, \alpha)$ with

$$\mathcal{P}_a : \left\{ \left\{ \forall n' \neq n \neq n_i, \forall r : \sum_{t':g(t')=n'} \tilde{s}_{t'r}^{(i)} < C_r \right\} \wedge \left\{ \forall r : \sum_{t':g(t')=n} \tilde{s}_{t'r}^{(i)} + \tilde{s}_{tr}^{(i)} < C_r \right\} \wedge \left\{ \forall r : \sum_{t':t'\neq t;g(t')=n_i} \tilde{s}_{t'r}^{(i)} - \tilde{s}_{tr}^{(i)} < C_r \right\} \right\}$$

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

Definition 9 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 $\sum_{i=1}^{NS} \tilde{\chi}(\mathcal{P}_a) \geq k(NS, 1-\varepsilon, \alpha)$ with

$$\mathcal{P}_a : \left\{ \left\{ \forall n' \neq g(t) \neq g(t'), \forall r : \sum_{t_1:g(t_1)=n'} \tilde{s}_{t_1r}^{(i)} < C_r \right\} \wedge \left\{ \forall r : \sum_{t_1:t_1\neq t;g(t_1)=g(t)} \tilde{s}_{t_1r}^{(i)} + \tilde{s}_{t'r}^{(i)} - \tilde{s}_{tr}^{(i)} < C_r \right\} \wedge \left\{ \forall r : \sum_{t_1:t_1\neq t';g(t_1)=g(t')} \tilde{s}_{t_1r}^{(i)} + \tilde{s}_{tr}^{(i)} - \tilde{s}_{t'r}^{(i)} < C_r \right\} \right\}$$

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

Besides these changes when defining the admissible neighbourhoods, the second pass of the GRASP 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.

Table 1 Grid instances

Inst.	#Vertices	#Nodes	C_n	Solutions (Sirdey 2011)
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

7 Computational experiments

7.1 Benchmark generation

7.1.1 Deterministic instances

In order to test our GRASP algorithm, we used several sets of test problems: grids to be placed on square grids, random data generated with TGFF[2] 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 of 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 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 torus architecture on which the vertices have to be placed. The results are given for a maximal bandwidth for the links of the different NoCs set to $B_a = 1{,}000$. The end column "Solutions" reports the solutions obtained by the semi-greedy algorithm for tasks mapping described in Sirdey (2011).

The real application we test here is a motion target application, which performs video processing and tracking of a sequence of related input video frames. 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 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 1,920 graphs with the number of vertices $|V|$ varying in $\{50, 100, 200\}$ 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 respectively 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

[2] Tasks Graph for Free: http://ziyang.eecs.umich.edu/~dickrp/tgff/.

$x \in \{1.01, 1.25, 1.5, 1.75, 2\}$. 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 $y = 5$.

7.1.2 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 considered the case when this ΣC application is composed of 57 tasks and has to be mapped on a Kalray architecture (Dupont de Dinechin et al. 2013), with a frequency of the chip of 400 MHz. 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 one time 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 $\%\,\text{ratio} = \frac{\text{computed rate}}{\text{max rate}} = 2.5 * 30 * p_{(s)} * 10^2$.

The experiments on grid instances 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.

7.2 Results for the deterministic version

Our GRASP algorithm was tested for different configurations, with $k \in \{2, 3, 4\}$ (see Algorithm 2, line 3), 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 min are reached.

Table 2 Results of GRASP method for grid problems

Name	k = 2			k = 3			k = 4		
	GR	PS-1	PS-2	GR	PS-1	PS-2	GR	PS-1	PS-2
Grid 4 × 4.inst	11	11	10	12	10	11	13	12	10
Grid 10 × 10.inst	73	69	69	75	70	69	76	69	69
Grid 12 × 12.inst	34	31	30	34	31	30	36	31	33
Grid 18 × 18.inst	92	86	88	91	91	91	99	98	91

Since we prioritize the minimization of the bandwidths (cf. Eq. 1) we guarantee just that this mapping is routable. As such, we are not guaranteeing an optimal routing and instead, we are analysing 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 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 $B_a = 1{,}000$ 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 neighbourhoods 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 Sirdey (2011). GRASP solutions have an average deviation from the solutions found by the semi-greedy method in Sirdey (2011) 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 target motion application, Table 3 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 "Sirdey 2011") for the placement of the application on a 2D torus 4×4 with $B_a = 10{,}000{,}000$. 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 $B_a = 100{,}000$. While none of the placements found by the method from Sirdey (2011) is routable

Table 3 Results for target motion application compared with greedy method from Sirdey (2011)

Name	ST	$\|V\|$	GRASP			Sirdey (2011)
			k = 2	k = 3	k = 4	
MD1.in	8	67	538,206	538,206	538,206	538,206
MD2.in	10	81	492,530	492,530	492,530	492,536
MD3.in	15	116	492,934	492,934	492,934	492,944
MD4.in	20	151	511,701	511,701	541,620	496,353
MD5.in	30	221	525,268	525,269	515,030	535,525
MD6.in	40	291	542,059	541,661	526,507	587,142

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 test 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, the higher the number of iterations is, the higher is the increase in the quality of solutions, with ≈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 1,113 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 Sirdey (2011). 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 1,920 instances), with 1,358 instances against 927 for the algorithm in Sirdey (2011).

The main reason of the relatively large number of instances for which our method does not find a solution is that we randomly generated instances which are very constrained (both in cluster capacity and in maximal bandwidth for each arc).

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. As for the value of the lb parameter, also an indicator of the quality of the routing, for 890 out of the 1,358 instances for which our GRASP is able to find a solution, this is equal to one, meaning that the routing is following a shortest path. For the remaining 468 solved instances, it seems that the routing is more difficult with lb value equal in average to 1.15.

Table 4 Computation results for $NS = 100$: grid problems

Instance	α	$\varepsilon = 0.05$			$\varepsilon = 0.1$		
		Sol.	C_n	Time	Sol.	C_n	Time
Grid 4×4	0.01	10	1.25	≈ 0	12	1.25	≈ 0
	0.05	10	1.25	≈ 0	12	1.25	≈ 0
Grid 10×10	0.01	74	1.15	0.92	76	1.15	0.56
	0.05	72	1.15	0.52	69	1.15	0.48
Grid 12×12	0.01	30	1	77	28	1	77.68
	0.05	32	1	58.5	28	1	67
Grid 18×18	0.01	92	1	600	89	1	600
	0.05	94	1	600	94	1	600

Table 5 Computation results for $NS = 1{,}000$: grid problems

Instance	α	$\varepsilon = 0.05$			$\varepsilon = 0.1$		
		Sol.	C_n	Time	Sol.	C_n	Time
Grid 4×4	0.01	10	1.25	≈ 0	12	1.25	≈ 0
	0.05	10	1.25	≈ 0	8	1.25	≈ 0
Grid 10×10	0.01	75	1.15	0.69	78	1.15	0.79
	0.05	76	1.15	0.72	79	1.15	0.95
Grid 12×12	0.01	29	1	230	31	1	210.6
	0.05	30	1	235.3	29	1	263
Grid 18×18	0.01	95	1	600	94	1	600
	0.05	91	1	600	93	1	600

7.3 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 or a time limit of 10 min 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 1,000. Tables 4 and 5 report the solutions obtained with column "sol." for the solution value, column "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 6 Computation time for $NS = 100$, $\varepsilon = 0.05$, $\alpha = 0.05$: TGFF problems

	$\|V\|$		
	50	100	200
# Instances	640	640	640
Time (s)	16.91	140.69	486.68

Table 7 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	NA
% Instances	68.33	13.91	1.46	1.09	2.81	12.4

As it can be seen, the quality of the solutions is consistent 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 instance, for "Grid 4×4" (respectively "Grid 10×10") it is necessary an increase of 1.25 (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 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 1,000 instead of a sample of 100 realizations. Like for the deterministic case, the execution time of the construction phase is largely superior to the one necessary for local search. By performing experiments for a sample size of 100, respectively 1,000, when $\alpha = 0.05$ and $\varepsilon = 0.1$, we found that the computation time for the construction step is in average $\approx$96 % of the overall execution time.

We also tested the algorithm on the 1,920 stochastic TGFF instances for a sample size of 100, when $\varepsilon = 0.95$ and $\alpha = 0.05$. Table 6 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.

As reported in Table 7, 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 $\approx$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.

There are 1,312 out of 1,920 total instances (68.33 %, see Table 7) for which our algorithm is able to find a feasible mapping without having to increase the capacity of the nodes. We did the comparison of the quality of solution for 1,297 of these instances for which the deterministic GRASP algorithm was also able to deliver a

Table 8 Quality of stochastic versus deterministic feasible solutions for same C_n: TGFF instances

	% Instances
$sol_s \leq sol_d + 5\,\%$	81.72
$sol_s \in]sol_d + 5\,\%, sol_d + 7\,\%]$	5.17
$sol_s \in]sol_d + 7\,\%; sol_d + 10\,\%]$	4.93
$sol_s > sol_d + 10\,\%$	8.17

feasible placement. The results are synthesized in Table 8 where the value of the stochastic solution sol_s is compared to the deterministic solution sol_d. For more than 81 % of the 1,297 instances, the value of the stochastic solution is at most 5% different from the one of the deterministic instance. For another ≈5 % of cases, the stochastic solutions are within 5 and 7 %, respectively within 7 and 10 %, of the values of their deterministic counterparts.

We can conclude thus that, in a vast majority of cases, it is possible to take into account the real variations of the parameters in order to have robust solutions without compromising too much of their quality.

8 Conclusion

In this paper we addressed the problem of joint placement and routing of dataflow applications on a clusterized architecture, for both deterministic and stochastic cases. In order to find routable placements, we have designed a GRASP for the deterministic version which was further adapted to the stochastic case. By using the robust binomial approach introduced in Stan et al. (2012), we solved the stochastic version under probabilistic constraints with uncertainty affecting the weights of the tasks. 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 1,920 synthetic graphs, for 25 % of cases, our GRASP method found mappings which are also routable while a sequential algorithm 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, within an acceptable execution time for our context application. Since the TGFF instances we use currently are very constrained and, thus there are a lot of cases for which a feasible routable mapping is not found (≈1/3 of cases), we plan in the near future extensible tests with other benchmarks.

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 showed that taking into account the variations of the weights of the tasks is particularly important in cases when the resources on the clusters are limited.

In the future, we plan to investigate several directions. The first one is to design a more efficient implementation of our GRASP by ameliorating the local search,

through the use of other neighbourhoods (e.g. cyclic exchanges). Also, the quality of the routing has to be further refined, one solution being to add in our problem model latency constraints. Finally, we have to complete our stochastic analysis by also taking into account the uncertainty affecting the bandwidths between tasks.

References

Aubry P, Beaucamps PE, Blanc F, Bodin B, Carpov S, Cudennec L, David V, Dore P, Dubrulle P, Dupont BdD, Galea F, Goubier T, Harrand M, Jones S, Lesage J, Louise S, Chaisemartin N, Nguyen T, Raynaud H, Sirdey R (2013) Extended cyclostatic dataflow program compilation and execution for an integrated manycore processor. In: Proceedings of the first international workshop on architecture, languages, compilation and hardware support for emerging manycore systems (ALCHEMY 2013), Barcelona, Spain, pp 1624–1633

Bonfietti A, Benini L, Lombardi M, Milano M (2010) 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. European Design and Automation Association, pp 897–902

Calafiore G, Campi M (2006) The scenario approach to robust control design. IEEE Trans Autom Control 51(5):742–753

Carpov S, Cudennec L, Sirdey R (2013) Throughput constrained parallelism reduction in cyclo-static dataflow applications. Procedia Comput Sci 18:30–39

Castrillon J, Tretter A, Leupers R, Ascheid G (2012) Communication-aware mapping of KPN applications onto heterogeneous mpsocs. In: Proceedings of the 49th annual design automation conference, DAC '12. ACM, New York, NY, pp 1266–1271

Choi J, Oh H, Kim S, Ha S (2012) Executing synchronous dataflow graphs on a SPM-based multicore architecture. In: Proceedings of the 49th annual design automation conference, DAC '12. ACM, New York, NY, pp 664–671

Dupont de Dinechin B, Guironnet de Massas G, Lager G, Léger C, Orgogozo B, Reybert J, Strudel T (2013) A distributed run-time environment for the Kalray MPPA-256 integrated manycore processor. Procedia Comput Sci 18(0):1654–1663

Feo T, Resende M (1995) Greedy randomized adaptive search procedures (GRASP). J Glob Optim 6: 109–133

Galea F, Sirdey R (2012) A parallel simulated annealing approach for the mapping of large process networks. In: IPDPS Workshop, pp 1787–1792

Garey M, Johnson D, Stockmeyer L (1976) Some simplified NP-complete graph problems. Theor Comput Sci 1(3):237–267

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

Hu J, Marculescu R (2005) Energy- and performance-aware mapping for regular NoC architectures. IEEE Trans Comput Aided Des Integr Circuits Syst 24(4):551–562

Korte B, Vygen J (2006) Combinatorial optimization: theory and algorithms, 3rd edn. Springer, Berlin

Lee E, Parks T (1995) Dataflow process networks. Proc IEEE 83(5):773–801

Lo VM (1988) Heuristic algorithms for task assignment in distributed systems. IEEE Trans Comput 37(11):1384–1397

Lombardi M, Milano M, Ruggiero M, Benini L (2010) Stochastic allocation and scheduling for conditional task graphs in multiprocessor systems-on-chip. J Sched 13:315–345

Manolache S, Eles P, Peng Z (2008) 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

Marcon C, Borin A, Susin A, Carro L, Wagner F (2005) Time and energy efficient mapping of embedded applications onto NoCs. In: ASP-DAC 2005, vol 1, pp 33–38

Marculescu R, Ogras U, Peh LS, Jerger N, Hoskote Y (2009) Outstanding research problems in NoC design: system, microarchitecture, and circuit perspectives. IEEE Trans Comput Aided Des Integr Circuits Syst 28(1):3–21

Marwedel P, Teich J, Kouveli G, Bacivarov I, Thiele L, Ha S, Lee C, Xu Q, Huang L (2011) 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. ACM, New York, NY, pp 109–118

Murali S, Benini L, De Micheli G (2006) A methodology for mapping multiple use-cases onto networks on Chips. In: DATE. IEEE, pp 118–123

Orsila H, Salminen E, Hämäläinen TD (2009) Parameterizing simulated annealing for distributing Kahn process networks on multiprocessor SoCs. In: Proceedings of the 11th international conference on System-on-chip, SOC'09, pp 19–26

Prékopa A (1995) Stochastic programming. Kluwer Academic Publishers, Dordrecht

Shestak V, Smith J, Uml R, Hale J, Moranville P, Maciejewski AA, Siegel HJ (2006) 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), pp 3–9

Singh AK, Shafique M, Kumar A, Henkel J (2013) Mapping on multi/many-core systems: survey of current and emerging trends. In: Proceedings of the 50th annual design automation conference, DAC '13. ACM, New York, NY, pp 1:1–1:10

Sirdey R (2011) Contributions á l'optimisation combinatoire pour l'embarqué: des autocommutateurs cellulaires aux microprocesseurs massivement paralléles. HDR, UTC, France

Srinivasan K, Chatha K (2005) A technique for low energy mapping and routing in Network-on-Chip architectures. In: ISLPED '05, pp 387–392

Stan O, Sirdey R, Carlier J, Nace D (2012) A heuristic algorithm for stochastic partitioning of process networks. In: Proceedings of the 16th IEEE international conference on system theory, control and computing (ICSTCC), pp 1–6

Stan O, Sirdey R, Carlier J, Nace D (2013) 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)

Stone H (1977) Multiprocessor scheduling with the aid of network flow algorithms. IEEE Trans Softw Eng 3(1):85–93