2013 Eighth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing

A GRASP for placement and routing of dataflow process networks on manycore architectures

Oana Stan*, Renaud Sirdey*, Jacques Carlier‡ and Dritan Nace‡
*CEA, LIST, Embedded Real Time System Laboratory,
Point Courrier 172, Gif-sur-Yvette, 91191 France.
Email: oana.stan@cea.fr, renaud.sirdey@cea.fr
‡UMR CNRS 6599 Heudiasyc, Université de Technologie de Compiègne
BP 20529, 60205 Compiègne, France.
Email: jacques.carlier@hds.utc.fr, d.nace@hds.utc.fr

***Abstract*—We propose a GRASP heuristic for solving the joint problem of placement and routing of process networks from the field of compilation for embedded manycore architectures. The method we propose consists in assigning applications expressed as dataflow process networks on homogeneous manycore architectures by taking into account the routing maximal capacity of the arcs for the underlying Network-On-Chip. Our experiments illustrate the algorithm ability to efficiently obtain good quality routable assignments, within an acceptable computational time, even for large size instances. Moreover, validation of our approach is also realized on data coming from an embedded application of video processing.**

***Keywords*-parallel embedded systems, dataflow compilation, placement, routing, GRASP heuristic**

I. Introduction

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. As such, the solution consists in designing and using parallel processing systems and nowadays a new generation of massively multicore microprocessors is emerging. The present multi-core architectures are achieving more performance by the use of several processing elements (roughly a dozen) and the next generation of manycore chips will be even more powerful, containing hundreds if not thousands of cores, connected via a Network-On-Chip (Noc). As such, we are entering into a manycore era in which the updated Moore's law states that the number of cores on a chip doubles every two years.

However, in order to efficiently exploit the parallelism and to take full advantage of the computing power these manycores may provide, their design requires new programming and execution paradigms as well as innovative compilation technologies. 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 first issue is already partially solved in the present embedded manycore like Kalray MPPA platforms [1] or Tilera due to some kind of hierarchical memory architecture, with distributed memories close to the processing elements and a shared on-chip memory for communication with other clusters and outside world.

Also, for taking advantage of manycores architectures in terms of computing power, power consumption or development cost, one must be able to efficiently parallelize an application and thus, it demands a "good" decomposition of the program into tasks. Traditional programming imperative languages (C or Java) are based on a sequential von Neumann architecture and therefore they are inappropriate for writing effective parallel programs. Dataflow paradigm seems to be a good candidate for programming manycore applications, which satisfy most of the properties stated before. With first models emerging in the early 1970s, dataflow languages provide an efficient and simple solution to express programs, which can be executed on a parallel architecture, without worrying about data synchronization.

An example of a recent dataflow model and language is ΣC (e.g. [2]) which provides more expressive power than Synchronous Dataflow Models (SDF) or Cyclo-static Dataflow models (CSDF), while allowing to perform a formal analysis for verifying properties like absence of deadlock or memory bounded execution.

In dataflow models, an application is described as a static instantiation graph of concurrent tasks 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. The *compilation process* for ΣC language is 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. These codes are intended either for off-line execution (the instantiation codes of the agents), either for further refinement, by substitution (the treatment codes corresponding to treatment agents).

- **Compilation of the parallelism.** The purpose of the second pass, the ΣC middle-end, is to instantiate and connect the agents, by executing at compile time the corresponding codes generated by the first pass. The C code, calling adapted APIs, is compiled and executed on the workstation for building the data structure representing the network of processes and generating the data for the agent instances.
  Once the construction of the application graph is complete, parallelism reduction techniques by pattern matching [3] are applied such as the application is compliant with the abstract specification of the system resources. During this stage, it is possible to verify the hierarchical coherence of the agents (for each subgraph verify that its composition implements correctly its state machine) and to perform a safe computation of a deadlock-free lowest bound for the buffers sizes of the links.
- **Resource allocation.** The third pass is in charge of resource allocation (in the larger sense). First, it supports a dimensioning of communication buffers taking into account the execution times of the tasks and the application requirements in terms of bandwidths (non functional constraints). Next, in order to realize a connection with the execution model, it constructs a folded (and thus, finitely representative) unbounded partial ordering of task occurrences.
  This pass is also responsible of placement and routing, with the objectives of grouping together (under capacity constraints for each cluster of the architecture) 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. Based on the partial orderings from the third pass, the runtime tables make the link with the execution model, by setting parameters of the system such as the configuration parameters of the NoC or the data structures describing the inter-task communication buffers. Also, during this stage and using C back-up compiler tools, link edition and loadbuild are realized.

As the above description shows, the compilation process of a dataflow application for a manycore architecture is becoming rather complex and requires solving a number of difficult and large-size optimization problems. Nowadays, the compiler design implies the application of operations research techniques not only to the so-called back-end (by solving more classical optimization problems, e.g. buffer sizing, instruction scheduling) but all along of the compilation process (e.g. process partitioning, quadratic assignment, multi-flow routing), in order to efficiently allocate and exploit the inter-related resources offered by parallel architectures.

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 as clusters communicating through an asynchronous Network-On-Chip. We propose a GRASP algorithm building solutions of high quality even for applications of relatively large instances in a reasonable computation time for our application context.

As shown in the sequel, even if the two sub-problems of tasks mapping and routing have already been addressed in the literature, the novelty of our approach consists in treating together task mapping and routing, and thus, taking into account the routing when placing the networks of processes, without any particular assumption on the network (here a Network-On-Chip) topology.

The rest of this paper is organized as follows. Section 2 gives a formal description of the problem and describes similar existing approaches. In Section 3, we present in detail the structure of our GRASP method. Section 4 provides the results of computational experiments conducted on synthetic benchmarks as well as on a real relatively complex ΣC application. Final remarks and some future perspectives are presented in Section 5.

II. The joint problem of placement and routing

A. Problem statement

In this paper, we study the static mapping of tasks from a DPN onto the network of clusters, such as the total bandwidth used by the application is minimal and for each pair of communicating tasks, there exits a routing path between tasks situated on different clusters.

The clusterized architecture is represented by a directed graph $G = (N, A, R, C_a)$ with N the set of nodes (clusters) and A the set of arcs between nodes, corresponding to the NoC links. $C_a : A \longrightarrow \mathbb{R}$ describes the bandwidths between different clusters of the target architecture, with $C_a((n_i n_j)) > 0$ the maximal capacity for arc (n_i, n_j) and $C_a((n_i n_j)) = 0$ if nodes (n_i, 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 $C_n \in \mathbb{R}^{+|R|}$.

Let $DPN = (V, E, S, Q)$ represent the network of processes with V the set of vertices (tasks) and E the set of communication channels. $S : V \longrightarrow \mathbb{R}^{+|R|}$, is a size function for the tasks, with s_{tr} being the weight of task t for resource r. The function $Q : E \longrightarrow \mathbb{R}$ characterizes the communication 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.

For the sake of simplicity, 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 C_a.

Let $g : V \to N$ 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:

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

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

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

Since the tasks mapping is equivalent to the Node Capacitated Graph Partitioning problem which is NP-hard [4] and unsplittable flow problem can be restricted to the Directed Edge Disjoint Paths problem, also NP-hard [5], the joint problem is straightforwardly NP-hard in the strong sense.

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 thousands 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 in the form of a bi-dimensional torus 4×4.

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 for embedded applications this algorithm is intended. The beginning of the development of an embedded application requires a short programmer/target feedback loop when the programmer is able to obtain a first working version of the application with a well coarse-grained structure. Thus, the beginning of the cycle requires for fast heuristics and can accept solutions of moderate quality. At the end of the development cycle, since more human and computing times are invested (e.g. acceptable compilation times to up of 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.

Also, other important factors which have to be taken into account into a mapping strategy are: the constraints coming from the platform and the application, the metrics to be optimized, the information and the assumptions made about the system, etc.

B. Algorithmic approaches

To the best of our knowledge, the joint problem of placement and routing of dataflow process networks on a homogeneous manycore architecture has not been yet addressed in the literature.

Even if the task mapping was and remains a relatively well studied problem (e.g. [6], [7]), the routing aspect was almost always neglected. Moreover, there are only a few approaches considering applications expressed as dataflow process networks and for which the target architecture is a manycore system. In [8], a simulated annealing algorithm is proposed for distributing Kahn Process Networks on Multiprocessors SoCs (MpSoCS) with at most four Processing Elements (PE) connected with dual shared bus. [9] proposes a parallel simulated approach for the DPNs mapping on a square torus architecture. Since this method is quite computationally demanding (roughly twenty minutes for a 31×31 square grid of tasks using 6 computing cores), it is more appropriate to be applied for the end of the development cycle of embedded applications.

Also, usually, the objective of existing techniques is a placement for which the tasks are equally distributed to all the processing elements (e.g.[10], [11]) while for our current approach, the interest is in minimizing the number of used clusters.

It is worth mentioning that the method proposed here does not belong to the family of static mappings for which the NoC is configured in function of the application in order to meet tasks requirements while fitting a specific SoC architecture (e.g. [12], [13], [14]). In [15], a branch-and-bound algorithm is proposed for the mapping of intellectual property blocks - IP (like CPU or DSP cores, video stream processors, input/output devices) on an architecture organized as regular tiles (composed of a processing core and a router), related by a NoC. The objective is to minimize the total energy spent on communication, by assuring that *no tile can host more than one IP* and having a routing constraint related to bandwidth usage. At each step, the algorithm assures a minimal and deadlock-free routing which respects the maximal load for each link of the NoC by incorporating a list of routing paths as part of the solution, instead of *a single routing path.* [14]

conceives dynamic re-configuration mechanisms to match the NoC configuration to the communication characteristics of each use-case. A design methodology, restricted to Æthereal NoCs, is introduced for mapping, path selection and resource reservation in the network, by taking as input use-cases of the SoC. The objective of the mapping process is to design the smallest size NoC, with the smallest number of switches which satisfies the constraints for all use-cases. Instead, we consider that the manycore specification and in particular NoC characteristics such maximal bandwidth for links are rigid and the placement and routing of tasks is realized afterwards (without worrying about the scheduling) during the compilation process of an application.

Between the only approaches similar to ours, of which we are aware of, is [16] treating the problem as a master (placement)/slave(routing) couple. As such, the overall problem is split into two sub-problems, less complex. The assignment is solved using a semi-greedy algorithm while the routing paths are computed optimally with a mixed linear integer programming. However, the sequential resolution can un-structure the initial problem and the found placement may not be routable and 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 in a placement non routable we cannot route because the flows between the nodes of the network exceed the maximal bandwidth capacity for the links C_a.

Other algorithmic aspects to be considered are the problem complexity and the size of instances we have to deal with, both factors making the building of a tractable exact resolution for both mapping and routing difficult and inappropriate. As such, we turned our attention to approximate algorithms and in particular to the GRASP metaheuristic, which seems a more suited choice to tackle this problem especially for the beginning of the development cycle of an application.

III. GRASP FOR THE JOINT PROBLEM OF PLACEMENT AND ROUTING

Introduced in the nineties by Feo and Resende [17], GRASP (Greedy Randomized Adaptative Search Procedure) is a multi-start metaheuristic, each iteration involving two phases: construction and local search. The construction phase builds a feasible solution using a greedy randomized algorithm. During the local phase, the neighborhood of the current solution is investigated in the search of better solutions. At the end, the best overall solution is kept as the result. Alg. 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

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 $g_b$
```

g_c found by the construction phase is further exploited in local search phase and optimized. If the resulting mapping g of this post-optimization is better than the previous best mapping g_b then we update g_b. Before explaining in more details each one of the two stages of our approach, let us recall the notions of total and relative affinity, initially introduced in [16] (see also [18] for details).

A. Preliminaries

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. Construction phase

Our greedy constructive method is an adaptation of an existing algorithm, initially used for partitioning networks of processes and based on the notion of relative affinity ([16], [18]). 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. The shortest path for flow f in the graph G' is denoted $sp(f)$. Initially, $A' = A$ and $\forall a \in A'$, $C_{r_a} = C_a$ 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 redefine the notions of *admissible assignment* and *admissible fusion* (see [18] for details), which for our approach, verify not only the respect of capacity resources but also the existence of a routing.

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

Definition 4: An assignment of task t to node n is admissible if it satisfies the capacity constraints for node n:

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

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

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

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

Definition 5: A fusion between the nodes n and m is admissible if:

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

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

After each assignment or fusion, C_{r_a} and F are updated accordingly, by adding or removing flows. The overall framework of the greedy randomized construction algorithm is presented in Alg.2. Initially, a partial solution is set as the first $\min(|V|, |N|)$ tasks in lexicographic order assigned to the N nodes with the condition that this initial mapping is also routable. Then, the list $[rcl]$ of k best decisions is constructed in a greedy fashion, by choosing between an admissible assignment or an admissible fusion, the one with the highest relative 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. If c_i is an assignment of task t_i to node n, the set W is updated: $W = W \setminus \{t_i\}$, the incoming / outgoing flows between the task t_i and the other tasks already assigned are computed and added to the set F and the residual capacities of the arcs of the network are reduced accordingly. If c_i is a merge of two nodes ($n_1^* \in N, n_2^* \in N$), the necessary modifications are made such that all vertices from node n_1^* are transferred to node n_2^*, the flows of the tasks already assigned are updated for taking into account the fusion and the residual capacities of the arcs of G' are also recomputed.

Algorithm 2 GRASP for joint placement and routing: construction_phase

```
Input: $V$, $N$, $R$, $k$
1: Initialization of the set of unassigned tasks $W = V$ and update sets $W$, $F$
2: Assign the first $\min(|V|, |N|)$ vertices to the $|N|$ nodes and update $W$
3: Build the list of $k$ restricted candidate decisions $[rcl]$ made of admissible assignments (cf. Def.4) and admissible fusions (cf. Def.5)
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 $g_c(V)$
```

C. Local search phase

Afterwards, the quality of the constructed solution S for g_c, is improved through a local search procedure. The neighborhood structure is the classical 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 neighborhood is appropriate under the assumption of a relative homogeneity for the tasks weights. Also, we are using a first improving search strategy in which the current solution is replaced by the first better local solution.In practice, it has been observed that for many applications, quite often, both search strategies lead to same final solution, but with smaller computation times when a first improving strategy is used [17].

The subtlety of our approach consists in selecting the tasks to 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 [16], [18]).

An exchange of two tasks t and t' from node $g(t)$ to node $g(t')$ and vice versa is admissible only if the capacity constraints for the associated nodes are respected and the flows having as source or sink t and/or t' are still routable. Since, except for the exchanged tasks, all the others remain on the same nodes, the computation of the value for the solution s' corresponding to a 2-OPT neighborhood can be realized quickly based on s and the bandwidths of exchanged tasks. The new value of the solution when moving t to $g(t')$ and t' to $g(t)$ will be:

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

IV. Experimental results

All the experiments 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.

We have decided to stop our algorithm either when the maximal number of iterations, equal to $\max(100, |V|log|V|)$ (V is the number of tasks to allocate), or a time limit of 10 minutes are reached.

A. Benchmarks

For testing our GRASP algorithm, we used three sets of test problems: several grids to be placed on square grids, a modified version of Johnson instances [19] 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 examples of undirected DPNs grids, representative in size of our application context, with unitary weights for tasks and for the communication channels. Besides, these instances are easy to modify and we can use them to test different configurations.Table I shows the grids instances, with column "#Vertices" the number of vertices to be placed and column "#Nodes" the number of clusters for a homogeneous tore architecture on which the vertices have to be placed. The results are giving for a maximal bandwidth for the links of the different NoCs set to $C_a = 1000$. The end column "Sol." reports the solutions obtained by the semi-greedy algorithm for tasks mapping described in [16]. The second set is composed of undirected graphs publicly available first used for bipartitioning [19], with different topologies and a number of vertices varying between 124 and 1000. We consider unitary weights for the channels between each communicating pair of vertices as well as unitary mono-dimensional weights for the vertices. The initial instances were adapted to be placed on a torus 2D of 4x4 nodes with maximal capacity on the arcs $C_a = 1000$.

Table I
Grid instances

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

Motion target: a real-case application

The real case we considered here for testing our method is a moving targets tracking application, with input a sequence of related video frames. Sequential video frames are analyzed and the movement of targets between the frames is outputted. Figure IV-A [16] gives an overview of the main components of the ΣC dataflow graph for this motion target application. The network, corresponding to an oriented graph, is described from left to right. The tasks io belonging to a special class of ΣC agents provide a way to handle the input/output. The left ones are in charge of reading two consecutive frames, of height H and width W and the right one is displaying the current image with the targets identified by surrounding rectangles. The next tasks S are data distribution agents of type $Split$ which take as input the current, and respectively, the previous image, and divide them into NB_S strips (into a round robin fashion). Production of these system agents is given by the NB_S parameter and the application size directly depends on this level of granularity, the further treatments being realized for each strip. As such, the following tasks Δ are computing the absolute difference for pixels by strips which is then used by σ to compute the standard deviation by macro-block and select the minimum for each strip. The outputs of Δ agents serve also as inputs for the t agents to construct a binary version of each strip which is further employed by the c tasks to detect connected components by strip. First m vertex representing a $subgraph$ (including a $join$ system agent and a user agent defining a median filter) compute the minimum deviation between all strips and broadcast the result to all strips. The second m vertex is another subgraph for merging together the bounding boxes found for each strip. The empty vertex is the Dup agent, used to duplicate data over all output channels. 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.

B. Results

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

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 analyzing the difference, for an obtained placement, between the routing our algorithm is using and an ideal one, (using a shortest-path strategy), by measuring the average for all flows $f \in F$ of fraction: $lb = \frac{length(sp(f))}{length^r}$ with $length^r$ being the shortest path in the NoC between $s(f)$ and $d(f)$.

Table II 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 $C_a = 1000$ and the number of selections $k \in \{2, 3, 4\}$. The column "GR" represents the results of the construction part while column "PS" presents the complete results with post-optimization, when 2OPT

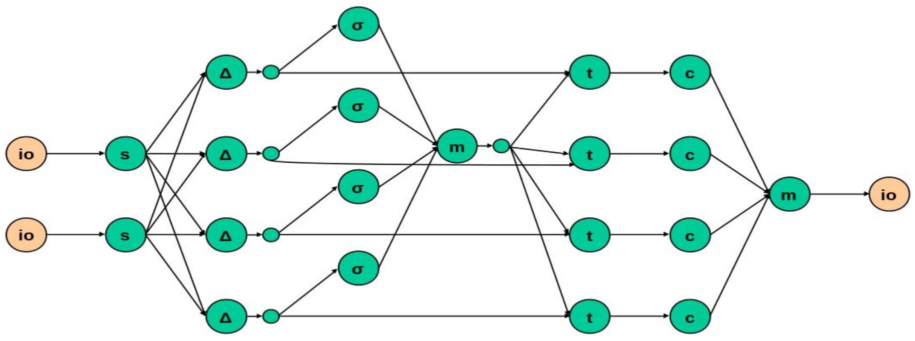

Figure 1. Motion target application.

neighborhood is 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 [16]. GRASP solutions have an average deviation from the solutions found by the semi-greedy method in [16] of $\approx$5% for $k = 2$ and less than 10% for $k = 3$ and $k = 4$, with the advantage that we also ensure the routability. When the capacity of arcs C_a is large enough, our method is able to accommodate the flows via the shortest paths and $lb = 1$ in all cases. Instead, when limiting more the capacity of the links, the average of lb tends to increase to 1.05. For the second set, as shown in

Table II
COMPUTATIONAL RESULTS OF GRASP METHOD FOR GRID PROBLEMS

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

table III the best results of our GRASP were obtained with the notion of relative affinity, when $k = 2$ and $k = 3$ with solutions of better quality than those found by [16], (reported in columns "Greedy") for 18 and respectively 17 instances (out of a total of 25). Also, the results are definitely better for $k = 2$ instead of $k = 4$ (for 20 out of 25 instances).

For the target motion application, Table IV 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, are compared with those obtained by the method currently implemented in the compilation chain (column "[16]") for the placement of the application on a 2D torus 4×4 with $C_a = 10000000$. The GRASP method provides better results in almost all cases. It should however be noted that when relative affinity is used instead, the results of the GRASP are of a lower quality. Since the capacity of the

Table III
COMPUTATIONAL RESULTS OF GRASP METHOD FOR JOHNSON INSTANCES COMPARED WITH GREEDY METHOD FROM [16]

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

network is large enough with regard to the flows to be routed, the bound $lb = 1$ for all instances, meaning that the routes found are following shortest paths.

The same instances were used for placing the target motion application on the same homogeneous NoC but this time with a maximal bandwidth for each arc $C_a = 100000$. While none of the placements found by the method from [16] is routable afterwards, the current method is finding placements which are also routable, with an average of 1.17 for lb.

V. CONCLUSION AND PERSPECTIVES

This paper addresses the joint placement and routing of networks of processes, arising in dataflow compilation for embedded manycore systems. The proposed GRASP

Table IV
COMPUTATIONAL RESULTS FOR TARGET MOTION APPLICATION COMPARED WITH GREEDY METHOD FROM [16]

Name	ST	$\|V\|$	GRASP			[16]
			k=2	k=3	k=4	
MD1.in	8	67	538206	538206	538206	538206
MD2.in	10	81	492530	492530	492530	492536
MD3.in	15	116	492934	492934	492934	492944
MD4.in	20	151	511701	511701	541620	496353
MD5.in	30	221	525268	525269	515030	535525
MD6.in	40	291	542059	541661	526507	587142

algorithm can solve, within an acceptable computation time (a dozen of minutes) and with a good solution quality, problems of relative large size (a few thousands of tasks on a dozen of nodes). As such, it could be used as an alternative to other tasks placement methods, at the beginning of the development cycle for embedded applications when the routing aspect is to be taken into account (for example several applications to be placed on the same manycore platform or complex applications with a high demand in computational power).

In the future, we plan to investigate several directions. The first one is to design a more efficient implementation of our local search, by investigating the effect of other neighborhoods (e.g. cyclic exchanges) which could provide solutions of better quality. A second direction could consist in improving the routing paths, by adding latency constraints for each pair of communicating tasks or by verifying that all the bandwidths between different clusters are accommodated via shortest-paths. Another perspective consists in extending our algorithm to cope with stochastic tasks weights which, because of their partial dependence on computing kernels execution times, are random variables.

REFERENCES

[1] B. Dupont de Dinechin, G. Guironnet de Massas, G. Lager, C. Léger, B. Orgogozo, J. Reybert, and T. Strudel, "A Distributed Run-Time Environment for the Kalray MPPA-256 Integrated Manycore Processor," *Procedia Computer Science*, vol. 18, no. 0, pp. 1654 – 1663, 2013.

[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] S. Carpov, L. Cudennec, and R. Sirdey, "Throughput constrained parallelism reduction in cyclo-static dataflow applications," *Procedia Computer Science*, vol. 18, no. 0, pp. 30–39, 2013.

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

[5] B. Korte and J. Vygen, *Combinatorial Optimization: Theory and Algorithms*. Springer, 3rd ed., 2006.

[6] V. M. Lo, "Heuristic algorithms for task assignment in distributed systems," *IEEE Trans. Comput.*, vol. 37, no. 11, pp. 1384–1397, 1988.

[7] H. Stone, "Multiprocessor scheduling with the aid of network flow algorithms," *IEEE Transactions on Software Engineering*, vol. 3, no. 1, pp. 85–93, 1977.

[8] H. Orsila, E. Salminen, and T. D. Hämäläinen, "Parameterizing simulated annealing for distributing Kahn process networks on multiprocessor SoCs," in *Proceedings of the 11th international conference on System-on-chip*, SOC'09, pp. 19–26, 2009.

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

[10] C. Walshaw, M. Cross, M. G. Everett, S. P. Johnson, and K. McManus, "Partitioning & mapping of unstructured meshes to parallel machine topologies," in *Procedeedings on Parallel Algorithms for Irregularly Structured Problems (IRREGULAR)*, pp. 121–126, 1995.

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

[12] C. Marcon, A. Borin, A. Susin, L. Carro, and F. Wagner, "Time and energy efficient mapping of embedded applications onto NoCs," in *ASP-DAC 2005.*, pp. 33 – 38 Vol. 1, 2005.

[13] K. Srinivasan and K. Chatha, "A technique for low energy mapping and routing in Network-on-Chip architectures," in *ISLPED '05*, pp. 387 – 392, aug. 2005.

[14] S. Murali, L. Benini, and G. De Micheli, "A Methodology for mapping multiple use-cases onto Networks on Chips," in *DATE*, pp. 118–123, IEEE, 2006.

[15] J. Hu and R. Marculescu, "Energy- and performance-aware mapping for regular NoC architectures," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 24, no. 4, pp. 551–562, 2005.

[16] R. Sirdey, *Contributions à l'optimisation combinatoire pour l'embarqué: des autocommutateurs cellulaires aux microprocesseurs massivement parallèles.* HDR, UTC, France, 2011.

[17] T. Feo and M. Resende, "Greedy randomized adaptive search procedures (GRASP)," *Journal of Global Optimization*, vol. 6, pp. 109–133, 1995.

[18] O. Stan, R. Sirdey, J. Carlier, and D. Nace, "A heuristic algorithm for stochastic partitioning of process networks," in *Proceedings of the 16th IEEE International Conference on System Theory, Control and Computing (ICSTCC)*, pp. 1–6, 2012.

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