Ant Colony Heuristic for Mapping and Scheduling Tasks and Communications on Heterogeneous Embedded Systems

Fabrizio Ferrandi, *Member, IEEE,* Pier Luca Lanzi, *Member, IEEE,* Christian Pilato, *Student, IEEE,* Donatella Sciuto, *Senior Member, IEEE,* Antonino Tumeo, *Student, IEEE*

***Abstract*—To exploit the power of modern heterogeneous multiprocessor embedded platforms on partitioned applications, the designer usually needs to efficiently map and schedule all the tasks and the communications of the application, respecting the constraints imposed by the target architecture. Since the problem is heavily constrained, common methods used to explore such design space usually fail, obtaining low-quality solutions.**

In this paper, we propose an *Ant Colony Optimization* (ACO) heuristic that, given a model of the target architecture and the application, efficiently executes both scheduling and mapping to optimize the application performance. We compare our approach with several other heuristics, including simulated annealing, tabu search and genetic algorithms, on the performance to reach the optimum value and on the potential to explore the design space. We show that our approach obtains better results than other heuristics by at least 16% in average, despite an overhead in execution time. Finally, we validate the approach by scheduling and mapping a JPEG encoder on a realistic target architecture.

***Index Terms*—Ant Colony Optimization, Mapping, Scheduling, Communications, Multiprocessors, FPGA.**

I. Introduction

HETEROGENEOUS multiprocessor architectures are the de-facto standard for embedded system design [1]. Today, to accelerate the different parts of the applications, they are usually composed of several general purpose, digital signal, application specific processors and reconfigurable devices (e.g., Field Programmable Gate Arrays - FPGAs), interconnected through various communication mechanisms.

When developing such embedded systems, the designer has to determine when (*scheduling*) and where (*mapping*) the groups of operations (i.e., the *tasks*) and the data transfers (i.e., the *communications*) should be executed, depending on a set of constraints and dependences, in order to optimize some design metrics, e.g., the program execution time.

Manuscript received July 22, 2009; received in revised form November 10, 2009; accepted January 15, 2010. Research partially funded by the European Community's Sixth Framework Programme, hArtes project. This paper was recommended by Associate Editor Petru Eles.

Fabrizio Ferrandi, Pier Luca Lanzi, Christian Pilato and Donatella Sciuto are with the Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy. Antonino Tumeo is with Pacific Northwest National Laboratory, 902 Battelle Blvd, 99352 Richland WA USA. The work was done while the last author was with the Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy. Send any comments to: *pilato@elet.polimi.it*.

Scheduling and mapping are strongly interdependent and *NP-complete* problems [2]. So, they cannot be efficiently solved with exact algorithms and heuristic methods, able to find good solutions in reasonable time, are usually preferred. Several works [3], [4], [5], [6], [7] have appeared in literature to approach the scheduling and mapping of tasks and communications. Among them, stochastic search methods [3], [4], [5], often bio-inspired, which explore the design space and exploit the feedback from previous executions, have been recently demonstrated successful. Nevertheless, these approaches usually focus separately on one of the aspects and fail in obtaining good overall solutions due to the very constrained design space. Moreover, the few formulations that try to solve simultaneously multiple problems, work well for small instances but loose effectiveness when the size of the design space grows. General approaches, able to efficiently generate high-quality solutions for complex applications on the new generation of heterogeneous embedded architectures are definitely required.

In this paper, we present an algorithm, based on *Ant Colony Optimization* (ACO) [8], that efficiently solves the scheduling and mapping of tasks and communications, to reduce the total execution time of the entire application given a model of the target architecture. Our approach, based on stochastic and heuristic principles, differs from previous works (e.g., [3], [4], [5]) since it is able to gradually construct multiple combinations of scheduling and mapping of tasks and communications, correct by construction, and searching around them, cutting out only the non-promising zones of the design space.

The main contributions of this work can be summarized as follows:

- it presents an ACO algorithm that reduces the execution time of the application by exploring different solutions for mapping and scheduling of tasks and communications;
- it proposes an approximation of this algorithm, introducing a multi-stage decision process which reduces the execution time of the exploration, maintaining a good correlation between the two problems.
- it compares the proposed variants also with common heuristics and a mathematical formulation, demonstrating its effectiveness to approach such complex exploration on both synthetic and real-life benchmarks.

Finally, we also validate the applicability of our approach by scheduling and mapping a JPEG encoder on a heterogeneous MPSoC developed on a FPGA prototyping platform [9].

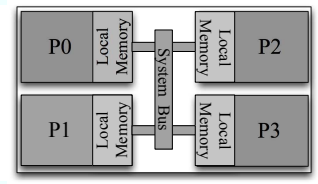

Fig. 1. The abstract model of a target architecture. Each component has 4 units of resource associated.

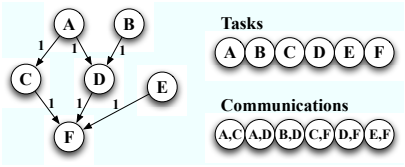

Fig. 2. Sample task graph. Edges are annotated with the amount of data to be transferred between source and target tasks.

The rest of the paper is organized as follows. In Section II, we define and formalize the problem that we address in this work. Section III discusses some background work, presenting and motivating the Ant Colony Optimization heuristic. Section IV details our formulation, that is then evaluated in Section V. Finally, Section VI concludes the paper.

II. Preliminaries

In this section, we present the basics of the problem that we address in this paper. In particular, we present the abstract model of the target architecture and the partitioned application. Then, we formalize the mapping and scheduling problem on these models.

A. Target Architecture

This work targets a general architectural model A for a heterogeneous Multi-Processor System-on-Chip (MPSoC), composed as follows:

$$A = P \cup C \tag{1}$$

where P is the set of processing elements (executing the different parts of the application) and C the set of communication components (performing the data transfers). A simple example is shown in Fig. 1 and it is composed of four processing elements, that communicate through a single system bus.

As in the formulation of the Multi-mode Resource Constrained Scheduling Problem (MRCSP) [10], each component of the architecture has a set $\mathcal{Q}$ of resources associated. These resources are then classified into two different classes: *renewable* resources $\mathcal{R}$, which return fully available after having been used, and *non-renewable* resources $\mathcal{N}$, for which the quantity consumed by the execution of a job cannot be replaced. For example, the area of hardware components (e.g., FPGAs) is considered *non-renewable* if the functionality cannot be reconfigured. The local memory of a processor is usually *renewable*, since it can be reused after a task has been completed. However, when specific allocation policies are adopted (e.g., static allocation), it can become *non-renewable*. In the example in Fig. 1, we assume that the processors $P0$, $P1$ and $P2$ have the *renewable* resources (e.g., data memories) q_0, q_1 and q_2, respectively. The resource q_3 of the processor $P3$ behaves as *non-renewable* (e.g., a memory with a static allocation policy). Therefore, each component $a_k \in A$ has associated, for each resource $q \in \mathcal{Q}$, a total amount of available resources A_k^q, that represents its *capacity* with respect to that resource. Tasks can be allocated on the components whose requirements of resources can be satisfied. In the given example, we can assume that 4 units are available for each resource on each component of the architecture.

Scheduling and mapping of an application should take into account additional details of the target architecture. First, we consider *resource sharing*, where a single implementation could be able to execute different instances inside the application. This happens, for example, when a hardware implementation is exploited by different tasks or when different software tasks exploit the same object code. In both cases, the resources (hardware area and instruction memory) are consumed only once. This requires to identify the tasks (or communications) that can share the implementations and correctly manage the constraints on the resources. Then, in this work we target architectures as the one shown in Fig. 1, where each processing element features local memories. In particular, when a task starts its execution on a processing element, the incoming data are read from the corresponding local memory. When the task ends, the produced data are transferred to the local memory of the processing elements of the successor tasks through a Direct Memory Access (DMA). As a consequence, data dependent tasks mapped on the same processor do not generate any communication overhead since a data transfer is not required, as in [11]. Otherwise, the communication is performed through one of the system busses and, through DMA, the communication may be overlapped with processing. In this way, only one actual data transfer is associated with each communication. However, the extension to other communication models [12] is straightforward. For example, with shared memories, two actual data transfers will be considered: from the source local memory to the shared memory and from the shared memory to the target local memory. In this work we assume that the communication model is the same for all the data transfers. We also assume that the execution time of the communication only depends on the quantity of data and on the performance of the component used for the communication and not on the processing elements that are involved.

B. Application Model

The mapping and scheduling problem requires to model the multi-task application to be executed on the target architecture as a Directed Acyclic Graph (DAG). A DAG is a graph $G = (T, E)$, without feedback edges, where vertices T represent groups of instructions (tasks) and the edges $E \subseteq T \times T$ the dependences among them. An edge $e(t', t) \in E$ implies that the task t can be executed only after the task t' and the data transfer associated with this edge have been completed. Each edge is also annotated with the amount of data exchanged from the source task to the target one. A simple example of such a task graph is shown in Figure 2. Similar graphs represent

data-flow dominant specifications that mainly refer to scientific and multimedia applications with high parallel computation on large blocks. In this case, their behavior can be statically predicted and, thus, the optimization can be performed off-line to efficiently reduce the application execution time. On the other hand, simple control constructs, such as function calls or countable loops, can be managed inlining the function or completely unrolling the loop body, respectively. This can greatly enlarge the application representation but allows to efficiently optimize, for example, the different iterations of one or more parallel loops [13]. Moreover, since large blocks have usually to be transferred from one stage to the other, efficient mapping and scheduling of communications have to be necessarily addressed.

Let $G = (T, E)$ be a DAG associated with an application and A the target architecture for its execution, which resources $\mathcal{Q}$ are classified in the set $\mathcal{R}$ of *renewable* and the set N of *non-renewable* ones.

A *job* j is defined as an activity to be performed on a component of the architecture. Thus, each task t is represented by a single job and, based on the adopted model of communication, one or more jobs are associated with each data transfer $e_{t',t}$. Thus, the entire application can be represented by a set J of jobs to be executed on the components A of the architecture.

An *implementation point* i is defined as a particular combination of resources and time required for the execution of a job j on a component a_k of the architecture. In fact, each job can have different implementations, not necessarily on all the components. For example, tasks cannot be assigned to communication components and vice-versa. Moreover, some processing elements could not be able to execute some tasks. For example, a task cannot be implemented in hardware if it contains constructs that cannot be synthesized. Different combinations of area and execution time are usually generated for hardware solutions. Software compilers can also produce different versions of the code, enabling different optimizations and resulting in different trade-offs between code size and performance. The set I contains all the implementation points available for the set of jobs J on the architecture A. Table I shows some examples of implementations. They represent the requirements in terms of time (i.e., clock cycles) and resources for all the tasks on the processing elements of the architecture. For each task, we have two different implementations on the component $P3$ (i.e., two different trade-offs for the requirement of the resource q_3).

The function $\gamma : I \rightarrow A$ returns the *component* corresponding to each implementation point. Note that, with this formulation, the constraint on the maximum number of components that can be used is satisfied by construction. In fact, it is not possible to assign a job to components that are not into the architecture, since the corresponding implementation points will not be generated. Different jobs can also share the same implementation point, modeling the *resource sharing*.

The function $\delta : I \rightarrow \mathbb{N}$ associates with each implementation point $i \in I$ the *execution time* for the related job on the associated component $a_k = \gamma(i)$.

The function $\sigma : I \times \mathcal{Q} \rightarrow \mathbb{N}$ associates with each implementation point $i \in I$ and resource $q \in \mathcal{Q}$ the *quantity of resource* required to implement the job on the admissible component $a_k = \gamma(i)$. Note that the implementation point i is generated only if the requirement can be satisfied by the component, i.e., $\sigma(i, q) \leq A_k^q$ where $a_k = \gamma(i)$.

TABLE I
DISTRIBUTION OF THE EXECUTION TIMES AND RESOURCE REQUIREMENTS FOR EACH IMPLEMENTATION POINT OF THE EXAMPLE TASK GRAPH. TWO DIFFERENT TRADE-OFFS ARE AVAILABLE FOR $P3$.

Task	P0		P1		P2		P3			
	$i_{t,0}$		$i_{t,1}$		$i_{t,2}$		$i_{t,3}$		$i_{t,4}$	
	time	q_0	time	q_1	time	q_2	time	q_3	time	q_3
A	8	2	2	3	6	4	1	2	3	1
B	5	3	10	4	6	3	2	2	4	1
C	4	4	6	2	4	1	1	2	3	1
D	8	1	2	2	7	3	3	1	1	3
E	10	3	3	1	8	3	1	3	2	1
F	3	4	7	3	2	2	1	2	2	1

These values, associated with each implementation point, can be obtained by estimation methods, by simulation, or by static or dynamic profiling the code of each task or communication on the target architecture.

C. Problem Definition

Let J, A and I be the set of jobs of the partitioned application, the abstract description of the target architecture and the set of all the available implementation points, respectively.

The *mapping* is defined by the function $M : J \rightarrow I$ that associates each job $j \in J$ with the proper implementation point $i \in I$ for the execution. On the other hand, the *scheduling* is defined by the function: $S : J \rightarrow \mathbb{N}$ that associates each job $j \in J$ with its start time.

In this paper, we focus on the optimization of the overall execution time of the application, that is the *make-span*. In particular, each job j, assigned to the implementation point $M(j) = i$, completes its execution at time:

$$H_j = S(j) + \delta(i) \tag{2}$$

The *make-span* Z is thus the overall execution time of all the jobs J on the architecture A, and it is defined as:

$$Z = max(H_j) \qquad \forall j \in J \tag{3}$$

Considering these equations, it is clear that the make-span depends on the start time of each job and the time spent for its execution on the component where it has been assigned. It is worth noting that other metrics (e.g., power consumption) can be optimized designing similar objective functions.

To reduce the make-span, the tasks should have been ideally assigned to the implementations where they complete with the minimum execution time. However, the solution has to satisfy additional constraints. First, the mapping is considered *feasible iff*:

$$\sum_{j \in J : a_k = \gamma(M(j))} \sigma(M(j), q) <= A_k^q \quad \forall a_k \in A, q \in \mathcal{N} \tag{4}$$

i.e., the requirements of *non-renewable* resources $\mathcal{N} \subseteq \mathcal{Q}$ should not exceed the available capacity on each component. Note that this equation also takes into account the resource sharing. In fact, if different tasks are assigned to the same

implementation point, the requirements of resources for the single implementation are considered only once.

On the other hand, the schedule is considered *valid* (i.e., the execution is correct) *iff* each job starts its execution only when its direct predecessors have been completed and when its component is free. Then, the following constraint has always to be satisfied:

$$max[H_{j'}, avail(a_k)] <= S(j) \quad \forall j' \in prec(j) : a_k = \gamma(M(j)) \tag{5}$$

where $S(j)$ represents the start time of the job j (assigned, through the mapping $i = M(j)$, to the component $a_k = \gamma(i)$), $H_{j'}$ the end time of the job j', $prec(j)$ contains all the directed predecessors of job j, and the function $avail(a_k)$ returns the time when the component a_k is available. This equation can also be rewritten as:

$$max[S(j') + \delta(M(j')), avail(a_k)] <= S(j) \quad \forall j' \in prec(j) : a_k = \gamma(M(j)) \tag{6}$$

where we underline that the end time of a predecessor job is strictly related to its mapping and its start time.

Equation 6 shows that the mapping and scheduling problems are strongly interdependent. In particular, the possibility of a job to start is strictly related not only to its own mapping, but also to the mapping of the jobs that have been executed before. For example, consider two jobs j' and j without any dependence, where j' has been already mapped and scheduled on component a'_k. If the job j is assigned to the same component ($a'_k = a_k$), it will be sequentially executed after the job j'. Instead, if the jobs are assigned to different components ($a'_k \neq a_k$), they can run in parallel. In this case, the order of execution of the (mapped) jobs can affect the availability of the resources and the quality of the results. Thus, mapping and scheduling have to be necessarily considered at the same time. In particular, the mapping should try to expose the maximum parallelism among the different jobs, limiting the contention on the resources, and different orderings for scheduling the jobs should be explored.

III. Related Work

Many different approaches on mapping, scheduling and communication synthesis have appeared in literature for the development of applications onto multi-processor embedded systems, with different models for the applications and formulations for the problems.

Besides DAGs, alternative models have been proposed. In particular, conditional task graphs have been introduced to optimize control-intensive applications and to exploit resource sharing [14] and voltage scaling [15] between mutually exclusive implementations. Since the behavior cannot be statically predicted, they usually attempt to optimize the average or worst-case execution time of the application, instead of the make-span. The proposed formulation can be easily adapted to this model, just by modifying the definition of the make-span and considering the mutual exclusion into the constraints for valid schedulings and feasible mappings. Cyclic or hierarchical task graphs [16] have been proposed to represent partitioned applications with feedback dependences. However, to determine an off-line schedule, the number of iterations has to be known in advance and, thus, DAGs can be obtained through loop-unrolling [13]. On the other hand, unrolling iterations introduces a large number of tasks into the representation and, for this reason, efficient and scalable methods for mapping and scheduling DAGs become crucial.

Scheduling and mapping approaches can be classified as on-line and off-line algorithms. In this work, we focus only on the latter since, with these approaches can obtain superior results, exploring a larger portion of the design space. Niemann and Marwedel [11] presented an Integer Linear Programming (ILP) formulation to derive the optimal solution for the mapping and scheduling problem on DAGs, considering heterogeneous architectures and communication costs. However, multiple implementations are considered only for hardware solutions and different communication models are not supported. Our formulation is thus more general and, considering *renewable* and *non-renewable* resources, we are able to approach a larger class of target platforms with different constraints. Furthermore, we consider multiple implementations also for software solutions, that is crucial when, for example, there are limits on the memory size. Unfortunately, mapping and scheduling an arbitrary DAG onto a system with limited resources is NP-complete and, thus, common approaches rely on heuristics to find near-optimal solutions in a reasonable time. Moreover, they often decompose the problem into sub-problems, i.e., separating the mapping from the scheduling. In particular, different algorithms aim to find the best start times for each one of tasks, which mapping is given. This problem has been widely studied [17] and, besides exact formulations [18] that are impracticable for large designs, list-based algorithms are usually adopted to determine a heuristic solution. These algorithms exploit a priority list to determine the order in which the operations are scheduled. Several methods are thus applied to explore only the scheduling by finding the best priority list, including optimization heuristics like Simulated Annealing (SA), Tabu Search (TS) [19] and Genetic Algorithms (GAs) [20]. We exploit the same concept to determine the priority values for the different jobs. It is proven [21] that these exploration algorithms, exploring different alternatives, outperform one-shot heuristics, despite a longer elaboration time. Thus, they are usually preferred when the scheduling can be performed off-line. Other algorithms, instead, explore only the mapping by determining the best processing elements for the tasks and evaluating each solution with a deterministic scheduling algorithm. Heuristic search methods, like GAs [3], TS and SA [4], [5], have been demonstrated to obtain better results also for this problem. The Kernighan-Lin-Fiduccia-Mattheyses (KLFM) heuristic has also been successfully adopted [22], but with higher complexity and execution times than the other methods. All these approaches, without considering the correlations, potentially lead to sub-optimal solutions and, when applied to hardly constrained design spaces, they can easily lead to constraint violations and unfeasible solutions. In general, returning in the space of the feasible solutions requires recovery mechanisms [23] that usually introduce a bias and limit the exploration.

Different methods have been exploited for the communications during the system-level synthesis [23], [24], [25] and design space exploration [26], [27]. Some works only attempt to minimize the transfers [4], [28] between the different groups of tasks, without considering bus contention. Other works exploit the communication synthesis during the definition of the architecture, usually attempting to meet the performance requirements by generating also complex communication infrastructures, if needed (e.g., [26], [27]) or by analyzing the communications independently from the synthesis of the components [29]. In our formulation, the bus contention has to be approached only with an efficient scheduling and mapping of the communications, since we cannot modify the architecture. Yen and Wolf [12] presented a classification of the different communication models and integrated their synthesis while defining the architecture, with relocation of tasks and communications on the different components. Differently from many existing works that focus on a single model (e.g., [4], [5], [23], [28]), we are able to support platforms with all the communication models presented in [12] and, through the *communication jobs*, effectively determine the communication configuration for the application. Moreover, few works (e.g., [24]) consider the resource requirements for the communication links and different implementations are usually not explored for the communications.

In conclusion, we definitely require constructive methods that are able to efficiently explore all the dimensions of the problems to obtain efficient implementations for the applications on a large class of target platforms.

A. Ant Colony Optimization

Ant Colony Optimization (ACO) is a modern technique, based on a stochastic decision process, originally introduced [8] for the Traveling Salesman Problem. It has been inspired by the cooperative behavior of ants when searching for food. In particular, all the ants start from their nest going in random directions, depositing a trail of *pheromone*. As time goes by, the shortest path to the food will contain more and more pheromones, motivating the other ants to follow this path instead of longer routes. The ACO heuristic is suitable for problems in which the solution can be found through subsequent decisions. The quantity of pheromone, stored into a matrix, represents the probability, for each decision, to lead to a good solution. All the decisions are initialized with a uniform probability. Then, iteratively, a certain number of routines (ants) are started to construct different solutions. At each decision point, a probability is generated for each of the admissible choices as follows:

$$p_{x,y} = \frac{[\tau_{x,y}]^{\alpha} * [\eta_{x,y}]^{\beta}}{\sum_{l \in \Omega_x} [\tau_{x,l}]^{\alpha} * [\eta_{x,l}]^{\beta}} \tag{7}$$

where x is the present point in the decision process, and y is the candidate destination, η is a problem-related local heuristic (i.e., calculated every time a probability is generated), while τ is the global heuristic, determined by the pheromone. Both contributions, weighted through α and β, influence the decision of the ant. Ω_x contains all the choices at the present point. At the end of each iteration, the results are ranked and the pheromones updated through different policies [30]. In general, the pheromones are updated as follows:

$$\tau_{x,y} = (1 - \rho) * \tau_{x,y} + \epsilon \tag{8}$$

where ρ is the *evaporation rate* (i.e., a parameter that controls how fast the pheromones are reduced) and $\epsilon \neq 0$ *iff* the decision is contained into the best solution. ϵ is a term usually proportional to the quality of the solution to maintain consistency among different iterations. In this way, only the best choices are reinforced and the others are penalized through evaporation. The best overall solution can be thus identified when the global heuristic will become dominant with respect to the local one. Moreover, since the probability in Eq. 7 is generated only for admissible choices, the algorithm is able to avoid the decisions that would violate a constraint, reducing the number of unfeasible solutions.

Recently, the ACO has been demonstrated superior to TS, GA and SA for both the standard [31] and the multi-mode [32] resource-constrained scheduling problem. Different works extended this formulation to the embedded systems design, considering mapping and scheduling separately [33], [34] or simultaneously [35]. However, these formulations are able to approach only specific sub-problems of our formulation. In fact, in [33], the authors propose algorithms for time- and resource-constrained scheduling for High Level Synthesis, that, exploiting the Max-Min update heuristic, determine the priorities for the scheduling or the resource allocation. They also discuss an extension for supporting multiple modes and constraints due to non-renewable resources, but communications are not considered. On the other hand, the same authors in [34] apply the methodology to assign application tasks to the processing elements of a heterogeneous multiprocessor with reconfigurable logic. However, their approach uses the ACO only for task mapping, and then schedules the resulting task graph with priority values obtained with a standard heuristic (i.e., mobility and total tardiness). This means that they explore different mapping solutions, but only one scheduling is obtained for each of them. In [35], the ACO aims at reducing the power consumption of the system, with a proper allocation of the tasks. However, all the components are considered as *renewable* and the constraints that may be imposed by the target architecture (i.e., area of the hardware devices) are not considered. Moreover, multiple implementations are not considered, as long as the communications. In conclusion, there is no formulation of ACO for the concurrent mapping and scheduling of heterogeneous embedded systems that is able to consider also communications and multiple implementations for each job to be performed.

IV. Proposed Methodology

In this section, we detail our ACO-based algorithm to perform the mapping and the scheduling of both tasks and communications on a heterogeneous MPSoC. First, we outline the overall methodology and we discuss how the concept of pheromone trails is applied to the specific problem. Then, we apply it to the illustrative example introduced in Section II and, finally, we discuss some problem-specific optimizations.

Algorithm 1 Pseudo-code of the proposed methodology.

```
1: generate initial solution
2: Z* ← Z_0
3: initialize pheromone values with 1/Z_0
4: for each ant l into the colony L do
5:    initialize candidate
6:    while candidate is not empty do
7:       select and assign job j to i
8:       update candidate
9:    end while
10:   estimate solution
11:   if Z_l < Z* then
12:      Z* ← Z_l
13:   end if
14: end for
15: if exploration is not terminated then
16:   update pheromone values
17:   perform random move with probability p_r
18:   return to 4
19: end if
20: return Z*
```

A. Methodology Overview

Our proposal separates the construction of the solution from its evaluation. In fact, to determine the effective start time of each task, we need to know when the incoming communications have been completed, if needed. However, due to the assumptions of the communication model, a communication between two data-dependent tasks is needed and thus it has to be analyzed only if the related source and target tasks have been assigned to different components. For this reason, in the first step, each ant gradually constructs the solution, choosing one job after the other, assigning it to a proper implementation point and analyzing the communications only after the related source and target tasks. Then, the evaluation of the obtained solution is based on the Serial Generation Scheme (SGS) schedule [31], that constructs a complete solution respecting the precedences and following a priority rule. In particular, the priority values correspond to the order in which the jobs have been selected by the ant, allowing to explore different scheduling solutions along with the job assignments to the implementation points. The pseudo-code of our formulation is described by Algorithm 1.

In detail, given the input task graph, the algorithm generates an initial solution (line **1**), for example, by assigning all the task jobs to the same component, provided that it is able to execute all of them, and all the communications to the associated local memory. It represents a fully-software feasible solution; its make-span Z_0 is used to initialize the current best solution (line **2**) and the pheromone values (line **3**), with an appropriate value ($1/Z_0$) to better scale the problem. After initializing the pheromones, the first colony of L ants is launched (line **4**). Each ant l is initialized (line **5**) with the jobs without predecessors as the set of initial candidates. From them, at each iteration, a job is selected and assigned to a proper implementation point (line **7**). Then, the candidate set can be updated with the jobs that have become eligible (line **8**), based on the job selected at line **7**. In particular, for communication jobs, the remaining jobs associated with the same data transfer are added, if any. On the other hand, for task jobs, the successor tasks become available, provided that all predecessors have been analyzed. Furthermore, as discussed above, the incoming communications can be also processed, if the related predecessor is assigned to a different component. The loop **6-9** is repeated until all the tasks and the communications have been assigned. The solution is evaluated (line **10**) and it replaces the current best one if it is improved (line **12**). Note that the methodology can be applied to other metrics just by designing a solution estimation consistent with the optimization criterion. It is only required to design local heuristics that efficiently lead the decision process. If the exploration is not terminated (e.g., the maximum number of generations or evaluations has not been reached), the pheromones are updated (line **16**) and a local search heuristic is applied (line **17**) to the best solution, to improve its optimization. In particular, changing the position of the jobs inside the *trace* results in different priority values for the scheduling. On the other hand, an unfeasible solution can be obtained by changing the mapping of the jobs. In any case, the current best solution is substituted only when the local search finds a better solution. Finally, a new ant colony is launched and, at the end of the exploration, the best overall solution is returned (line **20**).

B. Pheromones and Heuristics

In our work, the pheromones are stored into a matrix (i.e., the *pheromone structure*) which represents, for each possible step and for each combination of candidate jobs and corresponding admissible implementation points, the probability that this decision would lead to a good final solution. Let $|J|$ and $|I|$ be the number of jobs and the number of all the admissible implementation points for the job, respectively. Supposing that in the worst case all the jobs have to be analyzed, this matrix has a size of $|J| \times |J| \times |I|$ elements. We then adapted the Equation 7, considering, at each decision point d (line **7**), the probability to assign a candidate j to one of its implementation points i as follows:

$$p_{d,j,i} = \frac{[\tau_{d,j,i}]^{\alpha} * [\eta_{d,j,i}]^{\beta}}{\sum_{k,n}[\tau_{d,j^k,i^n}]^{\alpha} * [\eta_{d,j^k,i^n}]^{\beta}} \tag{9}$$

where η is a local heuristic, which suggests how good is to assign j to i, and τ is the global heuristic, which maintain information of the decisions taken by the previous ants. This value is normalized with the sum of the values for all admissible choices to give a probability. Then, a roulette wheel extraction is performed and, at each step, the result of the decision process is a job assigned to one of its admissible implementation points.

A good local heuristic suggests decisions when the global reinforcements are similar and, thus, drives the search to good solutions from the beginning, allowing faster convergence time. To compute the local heuristic $\eta_{d,j,i}$ we exploit information about the utilization of the resources. In particular, let

TABLE II
EVALUATION OF THE SOLUTION BASED ON THE PRIORITY VALUES GIVEN BY THE ANT EXPLORATION AND REPORTED INSIDE ROUND BRACKETS.

Step	Candidate set	Scheduled Task
i	$A(1)$ - $B(3)$ - $E(4)$	A
ii	$C(2)$ - $B(3)$ - $E(4)$ - $A, D(6)$	C
iii	$B(3)$ - $E(4)$ - $A, D(6)$ - $C, F(10)$	B
iv	$E(4)$ - $A, D(6)$ - $B, D(9)$ - $C, F(10)$	E
v	$A, D(6)$ - $E, F(8)$ - $B, D(9)$ - $C, F(10)$	A, D
vi	$E, F(8)$ - $B, D(9)$ - $C, F(10)$	E, F
vii	$B, D(9)$ - $C, F(10)$	B, D
viii	$D(5)$ - $C, F(10)$	D
ix	$C, F(10)$ - $D, F(11)$	C, F
x	$D, F(11)$	D, F
xi	$F(7)$	F

$avail^*(a_k)$ be the sum of execution times of the jobs assigned to a component a_k, and H_j^* the finishing time of j in this data-structure, we compute the local heuristic $\eta_{d,j,i}$ as follows:

$$\eta_{d,j,i} = \frac{1}{max[H_{j'}^*, avail^*(\gamma(i))] + \delta(i)} \tag{10}$$

where j' represents the predecessors of j and $\delta(i)$ the execution time of the job j if executed on the component $a_k = \gamma(i)$. This metric generates larger probabilities for the combinations of jobs and implementation points that should be able to complete their execution as soon as possible, given the estimated availability $avail^*(a_k)$ of the target resource a_k. Obviously, the information is not complete. In fact, since the communications between j' and j will be analyzed in the following, the related mapping and scheduling can lead to a very different solution. In that case, the final solution evaluation will penalize this decision, reducing the corresponding global heuristic and avoiding to take again this decision in the future.

Note that, if the job j cannot be assigned, at step d, to the implementation point i (i.e., the corresponding requirements of resources cannot be satisfied), that combination will not be considered in the roulette wheel (i.e., $p_{d,j,i} = 0$), avoiding to take decisions that would lead to an unfeasible solution.

After the ants of a colony have constructed their solutions, these are ranked. Then, the mapping and scheduling choices of the best solution of the colony, along with the current overall best solution of the optimization process, are reinforced. In particular, the pheromones are updated as $\tau_{d,j,i} = (1 - \rho) * \tau_{d,j,i} + \epsilon$, where ρ is the evaporation rate associated with the pheromone structure and $\epsilon = \rho * \frac{1}{Z^*}$ if the decision is contained into the best solution, having make-span Z^*, and $\epsilon = 0$ otherwise.

C. Illustrative Example

In this section, we apply our methodology to generate and evaluate an ant solution for the task graph in Figure 2 on the target architecture in Figure 1, based on the annotations in Table I. In this example, we assume, for simplicity, that (i) the communication delays equal to a single time unit (i.e., the bus takes one cycle to transfer a unit of data) and (ii) tasks do not share implementation points.

In our algorithm, the ant initializes the candidate set (line **5** of Algorithm 1) with the jobs that have no dependences (e.g, A, B and E as shown in Fig. 3). Then we have to select and assign a job to an implementation point for the execution.

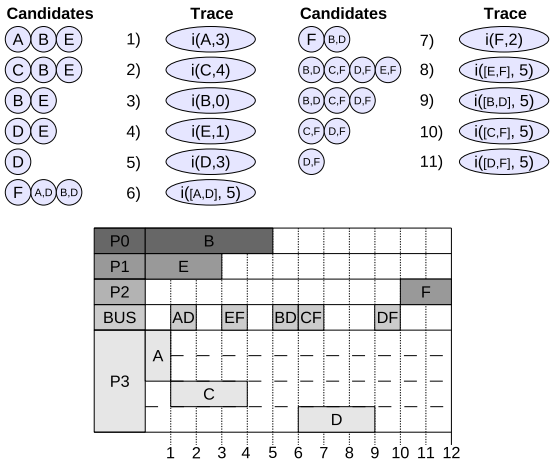

Fig. 3. Trace of the selection performed by the ant and the related schedule. Note that $i_{j,5}$ represents the implementation points on the system bus.

At the beginning the requirements for all the implementations can be satisfied by the components. Thus, at step 1, the ant can choose among 15 different combinations of jobs and implementation points, which probabilities are generated based on Equation 9. Since a roulette wheel selection is performed (line **7**), the selected combination is not necessarily the one with the highest probability. In our example, we assume that the ant selects *A* to be implemented on $i_{A,3}$ (i.e., the first implementation on $P3$) and thus the related resource is reduced to two units since it is *non-renewable*. No incoming communications are required by *A* and the candidate list is updated only with *C* that becomes available (line **8**). Now, only two units of the resource q_3 are available on $P3$. When, in step 2, the new probabilities are generated, the probability related to $i_{E,3}$, which requires 3 units, is not generated, avoiding the ant to select that combination. Let us assume that the ant selects *C*, assigning it to the implementation $i_{C,4}$. There is a communication among *A* and *C*, but these are both assigned to the same component. Therefore, this communication will be directly assigned to the associated local memory and not considered as a candidate job. At step 3, only *B* and *E* are available. There is only one unit of resource q_3 on $P3$, so only the implementations $i_{B,4}$ and $i_{E,4}$ are admissible on $P3$. The ant selects *B* on $i_{B,0}$ and, since *A* and *B* have been both analyzed, *D* becomes available. So, at step 4 the ant may choose among *D* and *E* with all the implementations for renewable resources (i.e., $i_{D,0}$, $i_{D,1}$, $i_{D,2}$, $i_{E,0}$, $i_{E,1}$ and $i_{E,2}$) and only with the implementations $i_{D,3}$ and $i_{E,4}$ for the resource q_3, i.e., the ones which requirements can be satisfied. The ant selects *E* and assigns it to $i_{E,1}$, and, at step 5, *D* is the only job in the candidate set. The ant assigns this job to the implementation $i_{D,3}$. *D* requires two data transfers, and consequently the two jobs (A, D) and (B, D) are added to the candidate list. Now the candidate set is composed of the jobs *F*, (A, D) and (B, D). The communication (A, D) is then selected at step 6 and task *F* is assigned to $i_{F,2}$ at step 7. At this point, (B, D), (C, F), (D, F) and (E, F) are available and, in the remaining steps, the ant selects (E, F), (B, D), (C, F) and (D, F), respectively.

The resulting trace is shown in Figure 3. Note that the step in which each job has been selected will correspond to the priority value for its scheduling. The associated make-span can be thus obtained by considering these priority values, as shown in Table II. In particular, at each step, the job with higher priority will be selected and scheduled. For example, starting from the first set of candidates, the task with higher priority is A, that starts at time 0 on $P3$. After executing A, the candidate set should be updated with the outcoming communications (A, C) and (A, D). However, A and C have been assigned to the same resource and the communication A, C is not needed. Instead, C is directly added to the candidate set, along with (A, D). This procedure is iteratively applied for all the jobs inside the application specification, until the candidate set is empty. The overall make-span is, thus, 12 time units long.

It is worth noting that different ant traces correspond to different ordering and thus different scheduling solutions. In fact, for example, if the ant had been selected (B, D) before (A, D) at step 6, *D* could have started only at time 7. In fact, (B, D) can start at time 5 and (A, D), that would have a lower priority, only at time 6, i.e., when the bus returns free. This shows how the scheduling of the communications can affect the final performance of the application and how the proposed approach can explore different combinations.

D. Problem Specific Optimizations

To reduce the memory requirements of the *pheromone structure* and, thus, the elaboration time, we also introduced a *two-stage decision process* for the ant. In particular, two matrices are created to store the pheromones and, at each scheduling step, two probabilities are calculated. Instead of performing the probability extraction on all the combinations of candidate jobs and the related implementation points, we initially allow the ant to select the job to schedule among the candidate ones, using the formula:

$$p^s_{d,j} = \frac{[\tau^s_{d,j}]^{\alpha^s} * [\eta^s_{d,j}]^{\beta^s}}{\sum_k [\tau^s_{d,j^k}]^{\alpha^s} * [\eta^s_{d,j^k}]^{\beta^s}}$$

which represents the probability that j is selected at the decision point d, where η^s and τ^s are the local and global heuristics specific for the scheduling problem, respectively. In our implementation, η^s is a linear combination of the mobility and the average execution time of the job on all the admissible target units. This provides a way to choose the jobs with lower mobility (i.e., higher impact on the critical path) or with a larger use of resources. The key idea is that, later, it is easier to find place for small or short jobs. In the same step, the ant decides the implementation point assigned to the selected job j with the formula:

$$p^m_{j,i} = \frac{[\tau^m_{j,i}]^{\alpha^m} * [\eta^m_{j,i}]^{\beta^m}}{\sum_n [\tau^m_{j,i^n}]^{\alpha^m} * [\eta^m_{j,i^n}]^{\beta^m}}$$

which expresses the probability to map j on the implementation point i. η^m and τ^m are the local and global heuristics specific for the mapping problem, respectively. η^m is a linear combination of the execution time of the job j on the implementation point i and a metric representing the global use of this component with respect to the number of candidate jobs.

The main advantage of this two-stage decision process is the reduction of the number of probabilities generated and the dimension of the pheromone structure. In fact, with this approach, two smaller pheromone structures are used. In particular, instead of the matrix of $|J| \times |J| \times |I|$ elements, two matrices are defined: one pheromone structure for the scheduling, of size $|J| \times |J|$, and one pheromone structure for the mapping, of size $|J| \times |I|$. The reduction of the complexity of the algorithm can also be verified in the previous example, where, for example, at step *i)* the number of probabilities is reduced from 15 to 8. In fact, 3 probabilities are generated to select the job inside the candidate set and, then, 5 mapping probabilities are generated for the selected job, one for each implementation point.

Furthermore, these two matrices are updated at the end of each generation with a different formula for each pheromone structure. In particular, the pheromones are updated for the scheduling as $\tau^s_{d,j} = (1-\rho^s) * \tau^s_{d,j} + \epsilon^s$ and for the mapping as $\tau^m_{j,i} = (1-\rho^m) * \tau^m_{j,i} + \epsilon^m$ where $\epsilon^s = \rho^s * \frac{1}{Z^*}$ and $\epsilon^m = \rho^m * \frac{1}{Z^*}$ if the decisions are contained into the best solution, which make-span is Z^*, and ρ^s, ρ^m are the evaporation rates for the two structures. Nevertheless, with this formulation, the information is spread into two different matrices and the correlation among the two decisions could be affected. This issue will be experimentally evaluated in the following section.

Finally, we added an enhancement, that is the *forgetting factor* [31], to reduce the possibility of converging to local minima. In particular, at the end of each colony, with a low probability, the heuristic substitutes the current best solution with the best trace of the current colony, even if it is worse. Obviously, the overall best is preserved, in case no other better solutions are found. The idea is that, if there are no other interesting points in the neighborhood of the current overall best, the algorithm reached a minimum and the evaporation rate slowly forces all the ants to converge to it. If it is only a local minimum, this may early cut out the possibility to find better solutions in other regions of the search space.

V. Experimental Evaluation

We implemented the methodology in C++ inside the PandA framework [36] and then evaluated our algorithm by applying it to several synthetic test cases and a real-life example (*smartphone*) on realistic models of target architectures. We compare our approaches (i.e., with one or two stages in the decision process) with three other common heuristics for the same problems in terms of time and number of evaluations requested to reach the optimum value, quality of the exploration results and overall execution time of the approaches. Finally, to validate the effectiveness of the methodology on real-world applications, we applied our algorithm to develop the JPEG encoder on a real platform.

A. Experimental Setup

We evaluated our approach on synthetic task graphs and on real-life benchmarks. In particular, we randomly generated several realistic task graphs using Task Graph For Free (TGFF) [37], which also allows the specification of a

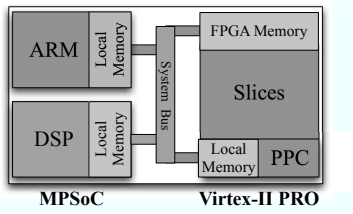

Fig. 4. The model of the target architecture $A1$.

model of the target architecture. In the first experiments, the architecture, namely $A1$ and shown in Fig. 4, is composed of four processing elements: a Digital Signal Processor (DSP), an ARM processor and a Virtex-II PRO XC2VP30 FPGA, that integrates a PowerPC (PPC) processor. We did not exploit partial dynamic reconfiguration and, thus, a task mapped on the FPGA cannot be removed. The area of the FPGA thus represents a *non-renewable* resource. The processing elements communicate through a DMA engine and, for the reconfigurable logic, we adopted a model similar to [6]. In particular, the tasks access a common memory (e.g., FPGAs internal memories - BRAMs) through an internal shared bus, which makes the access times independent of the placement of the tasks and negligible with respect to external transfers. Examples of similar platforms are the NXP Nexperia [38], the TI's OMAP [39] and the latest ATMEL DIOPSIS [40], adopted by several European Projects [41], [42] as target platforms with the same assumptions. For each task, performance annotations are generated on each component of the target architecture. In particular, each task takes 1,600$\pm$150 clock cycles on the ARM, 1,000$\pm$400 cycles on the DSP, 2,100$\pm$700 cycles on the PPC and 360$\pm$100 cycles on the FPGA. For the FPGA, which total available area has been configured to 15,360 slices (i.e., basic configurable elements of Xilinx FPGAs), each task occupies a different amount of logic elements based on the problem size (e.g., from 7,000$\pm$1,300 slices for smaller benchmarks to 500$\pm$50 slices for larger ones). Finally, each edge is annotated with a quantity of data (300$\pm$75) to be transferred.

We also applied our approach to a real-life example, that is the *smartphone* [43]. This benchmark is based on four publicly available applications: a GSM encoder/decoder, an MP3 decoder and a JPEG encoder. For the GSM and the MP3 applications, we target an architecture, namely $A2$, composed of 3 processors and 2 dedicated components, with realistic annotations for both tasks execution and data transfers [43]. Finally, for the JPEG encoder, we target a FPGA prototyping platform $A3$ [9] composed of one PPC, 3 MicroBlaze processors and an area dedicated to hardware accelerators. For this benchmark, an example of task graph is shown in Figure 5 and the related annotations, provided through profiling of the source code on the target platform, are reported in Table III. Each communication transfers the same quantity of data, so the related costs are fixed at 3,600,000 cycles per edge.

To compare the approaches proposed in this paper, we adapted some well-known heuristic methods to deal with multiple implementation points and with communication jobs.

Integer Linear Programming (ILP): we implemented a mathematical formulation that combines [10] and [11]. In particular, it deals with *renewable* and *non-renewable* resources instead of software and hardware ones and with the mapping on implementations rather than components. We used *Coin-Or* [44] to solve the instances associated with the benchmarks.

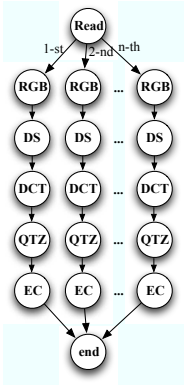

Fig. 5. Task graph of our JPEG parallel implementation.

TABLE III
PERFORMANCE OF THE JPEG TASKS ON THE RESOURCES AVAILABLE ON OUR TARGET PLATFORM $A3$.

Phase	PPC	MB	FPGA	
			Time	#Slices
Read	93,880,530	42,203,215	-	-
RGB	897,522	457,674	486,384	200
Downsampling (DS)	365,548	166,893	-	-
DCT	38,415,925	10,465,720	231,377	2,760
Quantization (QTZ)	47,603,256	4,388,594	-	-
EC	109,677,075	38,008,544	-	-

Ant Colony Optimization (ACO): this is the approach proposed in this paper and described in Section IV, both 1-stage and 2-stage. In the 1-stage process, $\alpha = \beta = 1$ (i.e., the weights for local and global heuristics) were used and the evaporation rate has been set to $\rho = 0.015$. In the 2-stage process, $\alpha^s = \alpha^m = \beta^s = \beta^m = 1$ were used along with different evaporation rates for the scheduling ($\rho^s = 0.025$) and the mapping ($\rho^m = 0.015$). The colony is composed of 10 ants for both the approaches.

Simulated Annealing (SA): the SA [4] is an adaptation from the *Neighborhood Search* (NS), a hill-climbing algorithm. Unlike NS, the SA can accept inferior solutions during its search according to a probability function. This probability starts high, and gradually drops as the temperature is reduced. When the temperature drops below a certain threshold, the algorithm ends. Among several cooling schedules, we adopted the geometric schedule ($T_{new} = \alpha T_{old}$). The initial temperature T_{start} is set to 250 and T_{finish} to 0.001 ($\alpha = 0.99$).

Tabu Search (TS): the TS [4] is another adaptation from the *Neighborhood Search*, that, instead, exploits a search history as a condition for the next moves. When generating new solutions, the TS checks its short term memory to avoid searching the same neighborhood (*tabu list*). The tabu list acts as a First In First Out (FIFO) queue: a set of freshly generated neighbours, not present in previous sets, is inserted in each iteration. However, to avoid stopping the search in promising area, after some time, the tabu list is released and the solutions become eligible again. We generate 10 neighbours with a tabu list composed of 5 sets of solutions each.

TABLE IV
COMPARISON ON THE EXECUTION TIMES (*time*) AND THE NUMBER OF EVALUATIONS (*#eval*) OF THE SEARCH METHODS TO REACH THE OPTIMUM VALUE (*ILP Opt.*). ILP HAS A TIME LIMIT OF 12 HOURS. WHEN THE ILP RESULT HAS NOT BEEN OBTAINED, THE AVERAGE OF THE BEST VALUES ARE REPORTED, ALONG WITH THEIR PERCENTAGE RELATIVE STANDARD DEVIATION. NUMBER OF TASKS (*#Tasks*) AND EDGES (*#Edges*) HAS REPORTED FOR EACH BENCHMARK. *Degree* REPRESENTS THE MAXIMUM NUMBER OF INCIDENT EDGES TO EACH TASK.

Bench	#Tasks/ #Edges	#Degree	ILP		ACO				SA		TS		GA	
					2-stage		1-stage							
			Opt.	time (s)	time (s)	#eval.	time (s)	#eval.	time (s)	#eval.	time (s)	#eval.	time (s)	#eval.
S1	5/4	2	3,027	0.22	0.20	365	**0.15**	**318**	0.23	4,721	0.36	2,555	0.19	1,020
S2	5/4	3	2,622	0.61	0,72	1269	**0.19**	**383**	0.61	12,911	1.04	7,077	1.05	5,235
S3	5/5	4	3,167	0.32	**0.10**	**188**	0.12	239	0.18	4,048	0.36	2,592	0.35	1,820
S4	10/9	2	4,525	17.41	**2.10**	**2,787**	2.90	2,931	(5,747 ± 6.25%)		20.12	80,887	6.47	23,172
S5	10/13	3	5,695	32.59	**3.36**	**3,018**	4.70	4,631	(6,814 ± 2.34%)		10.25	41,497	(6,041 ± 2.30%)	
S6	10/12	4	5,644	8.32	**2.82**	**2,700**	4.12	4,100	(6,458 ± 2.48%)		6.87	25,093	3.33	10,798
S7	15/16	2	7,318	7,152.89	(7,830 ± 0.97%)		(**7,491 ± 1.91%**)		(11,338 ± 8.12%)		(8,366 ± 3.23%)		(7,957 ± 1.71%)	
S8	15/22	3	8,358	711.78	(**8,814 ± 0.22%**)		(8,905 ± 2.51%)		(12,776 ± 2.50%)		(9,220 ± 3.75%)		(8,884 ± 1.81%)	
S9	15/26	4	9,618*	29,766.76	(10,120 ± 1.31%)		33.51	**12,505**	(14,237 ± 5.75%)		**9.17**	17,277	(10,293 ± 2.27%)	
S10	20/22	2	9,289*	24,275.33	(9,497 ± 0.13%)		(**9,490 ± 1.91%**)		(13,962 ± 0.29%)		(9,587 ± 0.98%)		(9,800 ± 4.27%)	
S11	20/32	3	9,530*	24,398.21	(**9,795 ± 2.01%**)		(10,019 ± 1.42%)		(15,089 ± 4.63%)		(10,326 ± 2.97%)		(10,666 ± 2.60%))	
S12	20/30	4	11,446*	24,297.01	**30.82**	**12,002**	(11,983 ± 2.02%)		(17,387 ± 4.93%)		(11,756 ± 1.28%)		(11,956 ± 2.60%)	

(* heuristic values, not demonstrated to be the optimum)

Genetic Algorithm (GA): a genetic algorithm [45] similar to [23] has been implemented using the *Open Beagle* framework [46]. In our formulation, the *crossover operator* combines two *parents* solutions into an offspring one, with a certain probability P_c (set to 0.70). Basically, crossover aims at exploiting the best features of two existing solutions to generate a new good solution. For the mapping, crossover is implemented as a standard single point crossover, that mixes the bindings of the jobs, and as a one-point topological crossover for what concerns the list of priorities. The *mutation operator* explores the design space around an existing solution. When applied to a parent solution, it generates an offspring solution introducing small random changes in the solution encoding, with a uniform probability. In particular, the priorities are modified with a variant of the *shift mutation* for the permutation representation, while the mapping assigns the jobs to a different implementation point among the available ones. The population is composed by 100 design solutions.

All these approaches feature local searches of the current solution. The SA and the TS are based on this concept to perform the exploration, but also the ACO (during the local optimization of the best solution) and the GA (in the mutation operator) can perform them. Several heuristics can be applied, in particular, to identify which jobs to change [47]. Since we are interested in the comparison of the search methods, we decided to apply the same basic random swap to all the approaches. This avoids a bias in the solutions and, anyhow, the improvements that can be obtained with other heuristics would affect in the same way all the approaches.

For all the explorations, we averaged the results over 30 runs of each task graph on a Intel Xeon X5355 (2.66 GHz and 8 MB of L2 cache) with 8 GB RAM.

B. Results

In the first experiment, we compared the search methods on the execution time and the number of evaluations required to reach the optimum value. We generated 12 relatively small benchmarks, namely S1-S12 in Table IV, varying the number of tasks and the maximum number of incident edges (*Degree*). Then, we computed the optimum value (*ILP Opt.*) with the ILP formulation with a time constraint of 12 hours. When the problem size grows over the 30 jobs (S9-S12), ILP reaches a value that is not proven to be optimal and thus we reported the obtained values. Finally, we applied the different exploration methods to these benchmarks and, for each of them, we measured the time (*time*) and the number of evaluations (*#eval*) needed to reach the value computed by ILP. Average results, along with Relative Standard Deviation (RSD) (i.e., the dispersion of the results obtained in the different runs from the average), are reported when the heuristics were not able to reach these values within 100,000 evaluations.

TABLE V
DIMENSION OF THE PROBLEMS AND THE ILP FORMULATIONS.

Bench	#Jobs	#Tasks/#Edges	#Variables	#Constraints
L1	22	10/12	332	833
L2	50	20/30	1,364	3,585
L3	130	50/80	10,409	28,363
L4	212	75/137	25,830	68,319
L5	268	100/168	41,856	113,221
L6	519	200/319	166,750	465,162
L7	768	300/468	370,873	1,047,848
L8	1,258	500/758	1,042,958	2,961,318
L9	1,868	750/1118	2,364,976	6,746,779
mp3	32	16/16	436	1,073
gsm-dec	90	34/56	3,703	10,039
gsm-enc	134	53/81	2,295	5,129
jpeg	101	51/50	6,281	26,239

The results in Table IV show that our approaches (*ACO 1-stage* and *ACO 2-stage*) can reach the optimum values much faster than the other heuristics, both in terms of execution time and number of evaluations. Moreover, both ACO variants reach the ILP values in more cases and perform better than the other search methods also when they are not able, due to the complexity of the exploration, to reach the ILP values (i.e., S7-S12). Moreover, the 2-stage ACO seems to scale better with respect to the 1-stage formulation when raising the size of the problem, both in terms of quality of the solutions and dispersion of the results. Comparing the other methods, the hill-climbing methods are usually very fast in small instances (as shown in particular by the SA results) but the number of iterations required to reach the ILP values are usually huge. In particular, the SA does not exploit any feedback and one provided by the TS (i.e., the tabu list) is not able to devise

TABLE VI
COMPARISON ON THE SEARCH METHODS ABOUT THE QUALITY OF THE RESULTS.

Bench	ACO			SA		TS		GA	
	2-stage	1-stage							
	Av. Best ± %RSD	Av. Best ± %RSD	(%diff)	Av. Best ± %RSD	(%diff)	Av. Best ± %RSD	(%diff)	Av. Best ± %RSD	(%diff)
L1	5,831 ± 4.48%	**5,705 ± 2.87%**	-2.16	7,702 ± 7.48%	+32.09	5,854 ± 3.25%	+0.39	5,832 ± 2.07%	+0.02
L2	12,312 ± 3.79%	**12,103 ± 3.08%**	-1.69	20,790 ± 6.28%	+68.86	12,634 ± 5.30%	+2.61	12,779 ± 3.93%	+3.80
L3	**29.598 ± 4.79%**	29,872 ± 3.15%	+0.97	66,053 ± 5.52%	+123.27	30,715 ± 7.85%	+3.82	35,445 ± 2.69%	+19.81
L4	**47,903 ± 5.40%**	49,395 ± 5.24%	+3.11	106,280 ± 5.11%	+121.86	54,746 ± 5.15%	+14.29	59,495 ± 3.27%	+24.20
L5	**60,163 ± 3.28%**	62,425 ± 4.12%	+3.76	127,248 ± 7.53%	+111.51	73,223 ± 2.54%	+21.71	77,333 ± 2.45%	+28.54
L6	**127,151 ± 3.87%**	136,991 ± 6.76%	+7.74	279,822 ± 4.76%	+120.07	168,690 ± 13.9%	+32.67	166,557 ± 2.02%	+30.99
L7	**195,897 ± 3.32%**	214,641 ± 6.58%	+9.54	438,762 ± 2.56%	+123.98	261,480 ± 5.37%	+33.48	249,584 ± 5.98%	+27.41
L8	**363,093 ± 4.03%**	408,765 ± 15.3%	+12.58	751,281 ± 1.96%	+106.91	465,115 ± 14.0%	+28.10	418,938 ± 4.26%	+15.38
L9	**521,399 ± 2.51%**	536,791 ± 1.94%	+2.95	1,124,864 ± 1.16%	+115.74	680,810 ± 0.96%	+30.57	620,041 ± 5.20%	+18.92
mp3	**62,164 ± 0.04%**	64,976 ± 0.01%	+4.52	113,243 ± 15.1%	+82.17	71,591 ± 10.4%	+15.17	65,292 ± 8.34%	+5.03
gsm-dec	**36,439 ± 1.92%**	38,523 ± 1.25%	+5.72	50,932 ± 8.41%	+39.77	46,424 ± 14.1%	+27.40	37,935 ± 2.68%	+4.11
gsm-enc	**92,384 ± 3.02%**	94,386 ± 3.24%	+2.17	170,133 ± 9.82%	+84.16	165,467 ± 18.5%	+79.11	106,888 ± 5.21%	+15.70
jpeg	**220,627K ± 4.37%**	251,610K ± 4.81%	+14.04	562,223K ± 9.50%	+154.83	238,721K ± 4.09%	+8.20	245,854K ± 4.93%	+11.43
		Avg.	+4.87		+98.97		+22.89		+15.80

TABLE VII
PERCENTAGE OF UNFEASIBLE SOLUTIONS GENERATED BY EACH SEARCH METHOD WHEN EVALUATING 25,000 SOLUTIONS.

Bench	ACO		SA	TS	GA
	2-stage	1-stage			
	(%)	(%)	(%)	(%)	(%)
L1	0.27	0.29	93.38	4.27	9.36
L2	0.30	0.33	97.92	6.57	15.44
L3	0.33	0.33	99.84	21.06	25.37
L4	0.32	0.32	99.84	20.78	27.97
L5	0.30	0.30	99.63	5.02	18.98
L6	0.32	0.34	99.58	30.82	12.86
L7	0.34	0.35	99.56	20.51	19.35
L8	0.35	0.35	99.52	52.30	32.22
L9	0.33	0.34	99.32	6.32	33.72
mp3	0.45	0.40	91.63	3.30	12.86
gsm-dec	0.34	0.27	95.77	4.74	17.50
gsm-enc	0.56	0.55	98.38	10.76	24.47
jpeg	0.10	0.08	98.92	1.95	7.33

TABLE VIII
COMPARISON AMONG THE OVERALL EXECUTION TIME OF THE SEARCH HEURISTICS, PERFORMING 25,000 EVALUATIONS.

Bench	ACO		SA	TS	GA
	2-stage	1-stage			
	time (s)	time (s)	time (s)	time (s)	time (s)
L1	8.40	12.38	0.88	4.41	9.95
L2	27.39	45.87	1.04	9.47	18.96
L3	117.02	251.43	1.52	28.23	48.22
L4	338.27	523.48	2.15	51.10	78.53
L5	479.37	764.89	3.06	86.26	119.37
L6	1,428.66	2,459.86	6.45	237.21	326.23
L7	2,883.99	5,178.30	10.58	606.39	541.28
L8	7,384.71	13,499.47	24.68	999.67	998.47
L9	14,021.67	26,385.01	65.93	4,044.71	1,918.59
mp3	22.39	23.18	0.91	5.21	8.24
gsm-dec	79.38	111.46	1.64	15.66	22.17
gsm-enc	86.33	108.46	1.49	19.90	27.70
jpeg	144.17	176.72	1.11	19.58	25.77

which are the sub-structures that can provide good solutions. On the opposite, the GA reaches the optimum values in the same time than the TS, but with a lower number of evaluations. In fact, the GA is able to identify and recombine good sub-structures, but the crossover and mutation operators require an additional execution time.

In the second experiment, we compared the same search heuristics with respect to the quality of the results on larger benchmarks (namely L1-L9 along with the four real-life applications) described in Table V. This table reports the size of the problems (i.e., number of jobs) and the related ILP formulations. In particular, due to the high number of variables and constraints, solving these problems optimally was not possible even increasing the time limit to 24 hours. The explorations have been thus executed until they reach the number of 25,000 evaluations since we observed that no improvements are usually obtained over that limit.

Table VI shows, for each one of the search algorithms, the average of the best solutions (*Av. Best*) obtained during the different runs, along with the RSD ($\%RSD$). The 2-stage ACO has been considered as reference point and, thus, percentage differences from its average best solutions have been also reported for the other heuristics. The results in Table VI show that the ACO methodologies outperform the other search methods in all the situations by at least 16% in average. In fact, the feedback obtained by the previous evaluations is more relevant than what reported by the other methods. In particular, the ACO suggests the better solutions as the GA, but also the better sequence of decisions to obtain that solution. Therefore, it is much easier to identify partial sub-structures that can contribute to the identification of a good solution. In addition, the 1-stage ACO better takes into account the correlation between mapping and scheduling problems, but only on small examples (e.g., $L1$ and $L2$). On the other hand, the 2-stage ACO scales better with problem size since the reduced number of probabilities that are generated allow the heuristics to be more significant during the exploration. The concept of population and ranking in GA (i.e., additional feedback to drive the exploration) mitigates the effect of random search, as shown by %RSD values in Table VI. Moreover, it seems to scale better with the size of the problems, with better results in larger examples. $L3$ has still a reasonable size to be approached with TS, but a structure difficult to be approached by the GA, that obtains poor results. In fact, when the GA has difficulties to identify the sub-structures, it usually obtains poor results. It is worth noting that the results obtained with real-life examples are consistent with the ones obtained with synthetic test cases of similar size. In particular, it results that our methodology behaves better than the other methods also approaching different architectural models.

Table VII reports the average number of unfeasible solutions that have been generated during the explorations. These results show that the constructive method of the ACO limits the number of unfeasible solutions, cutting out the unfeasible regions of the design space. The small percentage of unfeasible solutions are generated only by the local search heuristic implemented at the end of each colony. The SA obtains the worst performance in all the situations. In fact, random changes can easily produce unfeasible situations if specific conditions for the next moves (e.g., the tabu list into the TS) are not adopted. Since TS usually generates a limited set of unfeasible solutions, it is able to explore more solutions in the design space, obtaining better results. However, when it fails and gets stuck in sub-optima, the number of unfeasible solutions and the deviation of the results grow (e.g., $L6$ and $L8$). Even if the GA generates more unfeasible solutions than the TS, it is more robust than hill-climbing methods, as previous describe.

Finally, in Table VIII, we compare the average execution time (in seconds) of the different methodologies. It is worth noting that the overall execution time of the algorithms is composed of two main contributions: the time spent to generate the solutions and the time spent for their evaluation. In addition, the scheduler terminates the evaluation of a solution when it identifies a constraint violation. Thus, the SA has the shortest execution time with respect to the other algorithms since the time spent to evaluate the solutions is reduced (more than 96% of evaluations are terminated early). As discussed above, the ACO requires an additional time to build the solutions and almost all the solutions refer to feasible evaluations. Moreover, the 2-stage ACO generates a reduced number of probabilities at each decision point with respect to 1-stage ACO and thus it is able to contain also the execution time of the exploration.

C. Case study: JPEG

We applied our approach to the mapping of a real-life application, the JPEG compression algorithm. We target a heterogeneous multiprocessor prototype [9], developed on a Xilinx Virtex-II PRO XC2VP30 FPGA, that uses one of the two PPCs as a master processor and the other one as slave, along with the MicroBlaze (MB) soft cores. The master processor only manages tasks synchronization and sequencing on the slaves through interrupt signals. The slave processors have local memories, connected to a shared memory and to the master processor through a shared bus. Communications are performed through a DMA unit, which is controlled by the master processor and allows overlapping of data transfers and computations by the processing elements. The area on the FPGA not occupied by the soft cores and the other system components (i.e., buses, local and shared memory controllers, I/O controllers) can be used to implement hardware tasks. We configured different architectures, similar to the solution in Figure 1, where the slave PPC was supported by a varying number of slave MBs (from 1 to 3) integrating a Floating Point Unit (FPU), not available for the hard core. The presence of the FPU explains the better performance for the soft cores on arithmetic intensive tasks. The PPCs have a clock frequency of 200 MHz, while the rest of the system runs at 50 MHz.

Our JPEG implementation is divided into five phases: (1) RGB to YCbCr space color conversion, (2) Expansion and Down Sampling, (3) Bi-dimensional Discrete Cosine Transform (2D-DCT), (4) Quantization and (5) Entropic coding and file saving. These phases can be performed in separated chains for a minimum of 4 blocks of 8x8 pixels: the color space conversion operates per pixel, the 2D-DCT, the Quantization and the entropic coding works on single 8x8 blocks, while Down Sampling reaches optimal performance on 4 blocks, since a single value is averaged for each 4 chrominance values. It is thus possible to parallelize the application on data, extracting as many chains as desired depending on the size of the uncompressed image, as shown in Figure 5. The root task performs the image reading, while the end task is simply a stub. For our evaluation, we used images of different sizes and extracted task graphs with sizes varying from 20 to 50 tasks, thus presenting from 4 to 10 parallel chains, respectively. We had hardware implementations for the DCT and the RGB, and software implementations for the PPC and the MB for all the tasks. Table III reports the different execution times, in clock cycles, as seen from the master PPC, and the area, in slices, for the hardware cores. To simplify the scheduling of the data transfers, in these experiments, we do not exploit the support to the resource sharing for the hardware accelerators.

The first experiment compares the accuracy of the design solutions generated by the ACO approach for different task graphs on a platform with only one MB besides the slave PPC and thus the available area for the FPGA was of 8,400 slices. In Table IX, we compare the average results of the 2-stage ACO algorithm (*ACO*) with the performance of the resulting mapping on the platform (*Platform*) over 10 runs. The results show that the main differences (around 10%) are with 30 and 40 tasks. In fact, in these cases, we have a different distribution of the predicted and the real dynamic communication patterns, that are partially under-estimated by the ACO methodology. In particular, during the effective execution on the platform, the distribution of the communications often generates bus contention to be resolved by the interrupt controller. Therefore, the time spent by the interrupt to resolve these contentions generates that execution overhead that was not predicted by our communication model. On the other hand, with 20 and 50 tasks the predicted and the real patterns are more regular. In the former, the lower number of tasks and edges reduces the number of conflicts. In the latter, the execution of a higher number of parallel tasks masks the contentions on the bus and, thus, it results in a lower impact on the overall execution time. In conclusion, the ACO follows quite accurately the behavior of the platform and, in all the experiments, it correctly decided to implement three 2D-DCT in hardware. This shows that the methodology is able to identify efficient solutions for the mapping and scheduling problem.

In the second experiment, we executed the JPEG application composed of 50 tasks on different platforms, representing different combinations in the number of MicroBlaze processors (i.e., 3 downto 1) and the area for hardware implementations on the FPGA (i.e., 2,800 to 8,400). The results in Table X shows, as in the previous case, the prediction still remains quite accurate. In all cases, the heuristic is able to perceive,

TABLE IX
AVERAGE OF THE BEST SOLUTIONS OBTAINED BY ACO ALGORITHM AND ACTUAL PERFORMANCE ON THE TARGET PLATFORM WITH A VARYING NUMBER OF TASKS. THE ARCHITECTURE CONTAINS 1 POWER PC, 1 MICROBLAZES AND 8,400 FREE SLICES.

#Tasks/ #Edges	ACO		Platform		Diff.
	Avg.	%RSD	Avg.	%RSD	
20/20	170,416,083	2%	174,236,480	8%	2.19%
30/24	236,037,486	2%	268,569,432	7%	12.11%
40/32	324,599,846	1%	358,962,446	12%	9.57%
50/40	410,332,336	1%	427,313,801	7%	3.97%

TABLE X
AVERAGE OF THE BEST SOLUTIONS OBTAINED BY ACO ALGORITHM AND ACTUAL PERFORMANCE ON TARGET PLATFORMS WITH VARYING NUMBER OF SOFT CORES AND FREE AREA FOR HARDWARE ACCELERATORS. THE APPLICATION IS COMPOSED OF 50 TASKS.

MB/Area	ACO		Platform		Diff.
	Avg.	%RSD	Avg.	%RSD	
3/2,800	259,729,940	2%	248,492,658	15%	-4.52%
2/5,600	301,159,209	7%	283,861,438	11%	-6.09%
1/8,400	410,332,336	1%	427,313,801	7%	3.97%

mainly due to communication overhead, that it is better to use the hardware space to implement the 2D-DCT hardware accelerator instead of many, but slower, hardware RGBs. On the other hand, we also see that raising the number of processors and reducing the available area for hardware cores reduces the overall execution time of the program. In fact, the hardware core can execute only one task and, thus, more parallelism can be exploited with different general purpose processors that are able to execute all the tasks.

VI. CONCLUSION

In this paper, we described an ACO-based heuristic for mapping and scheduling both tasks and communications on heterogeneous multiprocessor architectures and we introduced a problem specific optimization, decoupling the choices for the scheduling and the mapping in a 2-stage decision process, that performs better when the size of the problems grows. We compared our ACO to previous heuristics and, considering the same number of evaluations, we obtained solutions 16% better in average, despite an overhead in execution time. Moreover, the proposed approach is able to reach the optimal solutions much faster than the other approaches. Finally, we applied our methodology to a real-world application on a realistic MPSoC. We showed that the ACO is able to produce efficient designs with a limited error in approximating the effective performance of the platform when implementing the related solutions.

REFERENCES

[1] W. Wolf, "The Future of Multiprocessor Systems-on-Chips," in *Proc. of 41st ACM/IEEE onf. on Design Automation (DAC '04)*, 2004, pp. 681–685.

[2] D. Bernstein, M. Rodeh, and I. Gertner, "On the Complexity of Scheduling Problems for Parallel/Pipelined Machines," *IEEE Transactions on Computers*, vol. 38, no. 9, pp. 1308–1313, 1989.

[3] J. I. Hidalgo and J. Lanchares, "Functional Partitioning for Hardware-Software Codesign Using Genetic Algorithms," *Proc. of the 23rd EUROMICRO Conference (EUROMICRO '97)*, pp. 631–638, 1997.

[4] T. Wiangtong, P. Cheung, and W. Luk, "Comparing Three Heuristic Search Methods for Functional Partitioning in Hardware–Software Codesign," *Design Automation for Embedded Systems*, vol. 6, no. 4, pp. 425–449, July 2002.

[5] P. Eles, Z. Peng, K. Kuchcinski, and A. Doboli, "System Level Hardware/Software Partitioning Based on Simulated Annealing and Tabu Search," *Design Automation for Embedded Systems*, vol. 2, no. 1, pp. 5–32, January 1997.

[6] S. Banerjee, E. Bozorgzadeh, and N. D. Dutt, "Integrating Physical Constraints in HW-SW Partitioning for Architectures With Partial Dynamic Reconfiguration," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 11, pp. 1189–1202, Nov. 2006.

[7] I. Issenin, E. Brockmeyer, B. Durinck, and N. D. Dutt, "Data-reuse-driven energy-aware cosynthesis of scratch pad memory and hierarchical bus-based communication architecture for multiprocessor streaming applications," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 8, pp. 1439–1452, Aug. 2008.

[8] M. Dorigo, V. Maniezzo, and A. Colorni, "The Ant System: Optimization by a colony of cooperating agents," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 26, no. 1, pp. 29–41, 1996.

[9] A. Tumeo, M. Branca, L. Camerini, M. Ceriani, M. Monchiero, G. Palermo, F. Ferrandi, and D. Sciuto, "Prototyping Pipelined Applications on a Heterogeneous FPGA Multiprocessor Virtual Platform," in *Proc. of IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC '09)*, Yokohama, Japan, Jan. 2009, pp. 317–322.

[10] P. Brucker, A. Drexl, R. Mohring, K. Neumann, and E. Pesch, "Resource-constrained project scheduling: Notation, classification, models, and methods," *European Journal of Operational Research*, vol. 112, no. 1, pp. 3–41, January 1999.

[11] R. Niemann and P. Marwedel, "An Algorithm for Hardware/Software Partitioning Using Mixed Integer Linear Programming," *Design Automation for Embedded Systems*, vol. 2, no. 2, pp. 125–163, March 1997.

[12] T.-Y. Yen and W. Wolf, "Communication synthesis for distributed embedded systems," in *Proc. of the 1995 IEEE/ACM International Conference on Computer-Aided Design (ICCAD '95)*. Washington, DC, USA: IEEE Computer Society, 1995, pp. 288–294.

[13] Y. Lam, J. Coutinho, W. Luk, and P. Leong, "Optimising multi-loop programs for heterogeneous computing systems," in *Proc. of the 5th Southern Conf. on Programmable Logic (SPL '09)*, 2009, pp. 129–134.

[14] Y. Xie and W. Wolf, "Allocation and scheduling of conditional task graph in hardware/software co-synthesis," in *Proc. of the conf. on Design, Automation and Test in Europe (DATE '01)*, 2001, pp. 620–625.

[15] D. Wu, B. Al-Hashimi, and P. Eles, "Scheduling and mapping of conditional task graphs for the synthesis of low power embedded systems," in *Proc. of Design, Automation and Test conf. in Europe (DATE '03)*, 2003, pp. 90–95.

[16] M. Girkar and C. D. Polychronopoulos, "The hierarchical task graph as a universal intermediate representation," *International Journal of Parallel Program*, vol. 22, no. 5, pp. 519–551, 1994.

[17] D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, K. C. Sevcik, and P. Wong, "Theory and Practice in Parallel Job Scheduling," in *Proc. of the Job Scheduling Strategies for Parallel Processing (IPPS '97)*, 1997, pp. 1–34.

[18] K. Wilken, J. Liu, and M. Heffernan, "Optimal instruction scheduling using integer programming," in *Proc. of the ACM conf. on Programming Language Design and Implementation (PLDI '00)*, 2000, pp. 121–133.

[19] S. J. Beaty, "Genetic Algorithms Versus Tabu Search for Instruction Scheduling," in *Proc. of International Conference on Neural Network and Genetic Algorithms*, February 1993, pp. 496–501.

[20] M. Grajcar, "Genetic list scheduling algorithm for scheduling and allocation on a loosely coupled heterogeneous multiprocessor system," in *Proc. of the 36th ACM/IEEE conference on Design Automation (DAC '99)*, 1999, pp. 280–285.

[21] S. Jin, G. Schiavone, and D. Turgut, "A performance study of multiprocessor task scheduling algorithms," *The Journal of Supercomputing*, vol. 43, pp. 77–97, 2008.

[22] "Extending the Kernighan/Lin Heuristic for Hardware and Software Functional Partitioning," *Design Automation for Embedded Systems*, vol. 2, no. 2, pp. 237–261, March 1997.

[23] J. Teich, T. Blickle, and L. Thiele, "An evolutionary approach to system-level synthesis," in *Proc. of the 5th International Workshop on Hardware/Software Co-Design (CODES '97)*, 1997, pp. 167–171.

[24] R. P. Dick and N. K. Jha, "Mogac: A multiobjective genetic algorithm for the co-synthesis of hardware-software embedded systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, pp. 920–935, 1997.

[25] G. Gogniat, M. Auguin, L. Bianco, and A. Pegatoquet, "Communication synthesis and hw/sw integration for embedded system design," in *Hardware/Software Codesign, 1998. (CODES/CASHE '98) Proc. of the Sixth International Workshop on*, Mar 1998, pp. 49–53.

[26] H. Nikolov, M. Thompson, T. Stefanov, A. Pimentel, S. Polstra, R. Bose, C. Zissulescu, and E. Deprettere, "Daedalus: Toward Composable Multimedia MP-SoC Design," in *Proc. of the 45th annual Design Automation Conference (DAC '08)*, 2008, pp. 574–579.
[27] C. Haubelt, T. Schlichter, J. Keinert, and M. Meredith, "SystemCoDesigner: automatic design space exploration and rapid prototyping from behavioral models," in *Proc. of the 45th annual Design Automation Conference (DAC '08)*, 2008, pp. 580–585.
[28] M. Purnaprajna, M. Reformat, and W. Pedrycz, "Genetic algorithms for hardware-software partitioning and optimal resource allocation," *Journal of System Architecture*, vol. 53, no. 7, pp. 339–354, 2007.
[29] S. Kim, C. Im, and S. Ha, "Efficient exploration of on-chip bus architectures and memory allocation," in *Proc. of the 2nd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis (CODES+ISSS '04)*, 2004, pp. 248–253.
[30] M. Dorigo and T. Stützle, *Ant Colony Optimization*. MIT press, 2004.
[31] D. Merkle, M. Middendorf, and H. Schmeck, "Ant colony optimization for resource-constrained project scheduling," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 4, pp. 333–346, Aug. 2002.
[32] C.-W. Chiang, Y.-Q. Huang, and W.-Y. Wang, "Ant colony optimization with parameter adaptation for multi-mode resource-constrained project scheduling," *J. Intell. Fuzzy Syst.*, vol. 19, no. 4,5, pp. 345–358, 2008.
[33] G. Wang, W. Gong, B. DeRenzi, and R. Kastner, "Ant Colony Optimizations for Resource- and Timing-Constrained Operation Scheduling," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 6, pp. 1010–1029, June 2007.
[34] G. Wang, W. Gong, B. DeRenzi, and R. Kastner, "Application partitioning on programmable platforms using the ant colony optimization," *Journal of Embedded Computing*, vol. 1, no. 12, pp. 1–18, 2005.
[35] P.-C. Chang, I.-W. Wu, J.-J. Shann, and C.-P. Chung, "ETAHM: An energy-aware task allocation algorithm for heterogeneous multiprocessor," *Proc. of the 45th ACM/IEEE conference on Design Automation (DAC '08)*, pp. 776–779, 2008.
[36] "PandA framework, http://trac.ws.dei.polimi.it/panda."
[37] R. P. Dick, D. L. Rhodes, and W. Wolf, "TGFF: task graphs for free," in *Proc. of the Sixth International Workshop on Hardware/Software Codesign (CODES/CASHE '98)*, Seattle, WA, Mar. 1998, pp. 97–101.
[38] "NXP. highly integrated, programmable system-on-chip (SoC). Available at http://www.nxp.com/products/nexperia."
[39] "Texas Instrument. OMAP Applications Processors. Available at http://dsp.ti.com." [Online]. Available: http://dsp.ti.com
[40] "ATMEL D940HF chip. Data sheet available at http://www.atmel.com."
[41] "hArtes project, ," Available at http://www.hartes.org.
[42] "MORPHEUS project, ," available at http://www.morpheus-ist.org.
[43] M. Schmitz, B. Al-Hashimi, and P. Eles, "Cosynthesis of energy-efficient multimode embedded systems with consideration of mode-execution probabilities," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 2, pp. 153–169, Feb. 2005.
[44] "COIN-OR (Common Infrastructure for Operations Research)," Available at `http://www.coin-or.org`.
[45] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, Mass.: Addison-Wesley, 1989.
[46] C. Gagné and M. Parizeau, "Open BEAGLE: A New Versatile C++ Framework for Evolutionary Computation." in *GECCO Late Breaking Papers*, 2002, pp. 161–168.
[47] Y. Lam, J. Coutinho, W. Luk, and P. Leong, "Mapping and scheduling with task clustering for heterogeneous computing systems," in *Proc. of Int. Conf. on Field Programmable Logic and Applications (FPL '08)*, Sept. 2008, pp. 275–280.

Fabrizio Ferrandi received his Laurea (cum laude) in Electronic Engineering in 1992 and the Ph.D. degree in Information and Automation Engineering (Computer Engineering) from the Politecnico di Milano, Italy, in 1997. He has been an Assistant Professor at the Politecnico di Milano, until 2002. Currently, he is an Associate Professor at the Dipartimento di Elettronica e Informazione of the Politecnico di Milano. His research interests include synthesis, verification simulation and testing of digital circuits and systems. Fabrizio Ferrandi is a Member of IEEE, of the IEEE Computer Society and of the Test Technology Technical Committee.

Pier Luca Lanzi was born in Turin, Italy, in 1967. He received the Laurea degree in computer science in 1994 from the Università degli Studi di Udine and the Ph.D. degree in Computer and Automation Engineering from the Politecnico di Milano in 1999. He is associate professor at the Politecnico di Milano, Dept. of Electronics and Information. His research areas include evolutionary computation, reinforcement learning, machine learning. He is interested in applications to data mining and computer games. He is member of the editorial board of the *Evolutionary Computation Journal*, the *IEEE Transaction on Computational Intelligence and AI in Games*, and *Evolutionary Intelligence*. He is also the editor in chief of the *SIGEVOlution*, the newsletter of the ACM Special Interest Group on Genetic and Evolutionary Computation.

Christian Pilato received his Laurea in Computer Engineering from Politecnico di Milano, Italy, in 2007. He is currently a third-year PhD student in Information Engineering at the same university. His research interests include high-level synthesis, evolutionary algorithms for design space exploration and multi-objective optimization, and multiprocessor designs.

Donatella Sciuto received her Laurea in Electronic Engineering from Politecnico di Milano and her PhD in Electrical and Computer Engineering from the University of Colorado, Boulder. She is currently a Full Professor at the Dipartimento di Elettronica e Informazione of the Politecnico di Milano, Italy. She is member IEEE, IFIP 10.5, EDAA. She is or has been member of different program committees of ACM and IEEE EDA conferences and workshops. Her main research interests cover the methodologies for the design of embedded systems and multicore systems, from the specification level down to the implementation of both the hardware and software components, including reconfigurable and adaptive systems. She has published over 200 papers. She has served as Associate Editor of the IEEE Transactions on Computers, and serves now as Associate Editor to the IEEE Embedded Systems Letters for the design methodologies topic area and as Associate Editor for the Journal of Design Automation of Embedded Systems, Springer. She has offered several technical services to IEEE: in particular she has been in the executive committee of DATE for the past ten years and she has been Technical Program Chair in 2006 and General Chair in 2008. She is General Co-Chair for 2009 and 2010 of ESWEEK. She has served also as executive committee member of ICCAD for three years and has served a 2 years term as VP of Finance for the Council of EDA, for which she serves as President elect for the next two years. She has received different IEEE service awards and the Outstanding Contribution Award from the Computer Society in 2009.

Antonino Tumeo received his Laurea in Information Engineering in 2005 and the PhD in Information and Automation Engineering in 2009 from Politecnico di Milano, Italy. His research interests include system level design and simulation of multi-threaded/multiprocessor and reconfigurable architectures, hardware-software codesign, FPGA prototyping and GPGPU computing. He is currently a Post Doc Research Associate in the High Performance Computing group at the Department of Energy's Pacific Northwest National Laboratory.