Ant Colony Optimization for Mapping and Scheduling in Heterogeneous Multiprocessor Systems

Antonino Tumeo, Christian Pilato, Fabrizio Ferrandi, Donatella Sciuto, Pier Luca Lanzi
Politecnico di Milano, Dipartimento di Elettronica e Informazione
Via Ponzio 34/5, Milano, Italy
{tumeo,pilato,ferrandi,sciuto,lanzi}@elet.polimi.it

***Abstract*—Heterogeneous multiprocessor systems, assembled with off-the-shelf processors and augmented with reprogrammable devices, thanks to their performance, cost effectiveness and flexibility, have become a standard platform for embedded systems. To fully exploit the computational power offered by these systems, great care should be taken when deciding on which processing element (*mapping*) and when (*scheduling*) executing the program tasks. Unfortunately, both these problems are *NP-complete*, and, even if they are strictly interconnected, they are normally performed separately with exact or heuristic algorithms to simplify the search for the optimum points.**

In this paper we present an exploration algorithm based on *Ant Colony Optimization* (ACO) that tries to solve the two problems simultaneously. We propose an implementation of the algorithm that gradually constructs feasible solution instances and searches around them rather than exploring a structure that already considers all the possible solutions. We introduce a two-stage decision mechanism that simplifies the data structures but lets the ant perform correlated choices for both the mapping and the scheduling. We show that this algorithm provides better and more robust solutions in less time than the Simulated Annealing and the Tabu Search algorithms, extended to support the combined scheduling and mapping problems. In particular, our ACO formulation can find, on average, solutions between 64% and 55% better than Simulated Annealing and Tabu Search.

I. Introduction

Efficiency is the key-word that designers have in mind while developing modern embedded systems. To cope with the strict application requirements, it is nowadays common to design these systems with several heterogeneous processing cores, which provide high performance on specific operations, while saving power. Beside General Purpose Processors (GPPs) and Digital Signal Processors (DSPs) for arithmetic intensive computations, they can also include configurable components, like Field Programmable Gate Arrays (FPGA), which allows a better customization for the target application.

When developing the software for these architectures, the programmer (or a semi-automatic tool) normally divides (*partitions*) the application in several different interdependent groups of operations (*tasks*), and, depending on their nature, assigns (*mapping*) them to the different processing elements. The partitioning phase identifies the parallelism among the tasks, that can in turn potentially run on several, if not all, processing elements of the system, but with different performance. Some of the tasks can also be implemented in hardware, using the configurable logic elements, occupying some of the available area. Aim of the developer is thus to find the best assignments to reduce the overall execution time, while providing the maximum efficiency and exploitation of the system and satisfying the given constraints in terms of processing elements and configurable logic area.

This assignment process, when many tasks and processing elements are involved, is usually complex, and requires a careful exploration of all the design space. Tasks are dependent on each other, meaning that, before executing some tasks, others have to finish their execution to provide input data. Therefore, the binding of a task to a resource heavily influences the *scheduling* of the tasks and, viceversa, a good scheduling could favour a better mapping of the tasks to the available resources. Furthermore, the goodness of the assignments can be correctly evaluated only after both mapping and scheduling have been performed. The common approach is to perform an initial mapping (i.e., deciding on which type of resource to launch the tasks) and then to execute a resource constrained scheduling to assess the quality of the overall process.

Nevertheless, both these problems are notoriously *NP*-complete [1]. This means that it is possible to reach optimal solutions through brute force search or Integer Linear Programming (ILP) formulations only for small instances of the problems, but it is usually preferred to provide suboptimal solutions with faster computation times. Several heuristic methods to solve these problems have been proposed and they include Simulated Annealing (SA), Tabu Search (TS) and Genetic Algorithms [2] (GA). Normally, after these iterative search methods have been applied to find a feasible mapping, eventually taking into account area constraints for the FPGA, the goodness of the generated solutions is evaluated applying classic scheduling algorithms (e.g., list based with simple heuristics, force directed, etc.). Techniques that apply a second pass with iterative search algorithms for the scheduling have also been proposed [3]. Recently, Ant Colony Optimization (ACO) [4] has emerged as an interesting technique to solve, separately, the mapping [5] or the resource constrained scheduling problems [6], [7]. This technique combines global and local heuristics to allow step-by-step decisions by a group of cooperating agents (the ants) and it seems particularly suitable to efficiently explore the search space of this class of

978-1-4244-1985-2/08/$25.00 ©2008 IEEE

problems that can be formulated as stochastic decision making processes, as well as the Traveling Salesman Problem (TSP) for which this method has been introduced. Our work focuses on the use of the ACO for the embedded system design field but, instead of dealing only with the mapping or the scheduling phase, it proposes a multi-modal approach to obtain a good *scheduled* and *mapped* task partitioning with area constraints for reconfigurable logic. The major contributions of this paper can be summarized as follows:

- it proposes an unified algorithm, based on the ACO technique, that simultaneously performs both the task allocation and scheduling of a task graph on a resource constrained, heterogeneous multiprocessor architecture with statically configurable devices;
- it proposes an ACO implementation that performs a step-by-step exploration of the search space, gradually constructing feasible solution instances, rather than exploring a structure that comprises all the solutions, using a two-stage decision mechanism;
- it extends two iterative techniques, SA and TS, commonly used to perform task mapping, to also search in the scheduling space, and it compares them with the proposed ACO based algorithm, proving its performance and robustness with respect to this class of problems.

The remainder of this paper is organized as follows: Section II defines the problem addressed, while Section III introduces the ACO heuristic. Section IV presents some of the related works in the area, and Section V illustrates the proposed mapping and scheduling algorithm based on ACO. Experimental evaluation and comparisons with SA and TS iterative techniques is proposed in Section VI. Finally, Section VII concludes the paper.

II. Problem Definition

Task mapping and scheduling are two important phases in the design of multiprocessor heterogeneous embedded systems. A task is a coarse grain set of instructions with a well defined interface which represents a part of a program. A program can be decomposed in several interdependent tasks, and can be represented through a Directed Acyclic Graph (DAG), the *task graph*. We formally define the task graph as $G =< T, E >$, where T is a set of vertices representing the tasks and E a set of directed edges that represents the dependencies among tasks. The relation $P : T \times M \rightarrow \mathbb{N}^+$ associates with each task t_i its *performance* on the processing element m_j, while the relation $A : T \rightarrow \mathbb{N}^+$ associates the *area* potentially occupied by the task t_i on the reconfigurable logic when it is mapped on the FPGA. Relations *P* and *A* can be considered as *task annotations*, and can be obtained by profiling the code of each task on the target processing elements (for performance annotations) or by estimation methods (for both performance and area annotations). In this formulation, a task, when executed on the system, can start only when all its predecessors have ended, runs only on a processing element and, after starting, cannot be interrupted. Fig. 1 shows a simple task graph with 6 nodes.

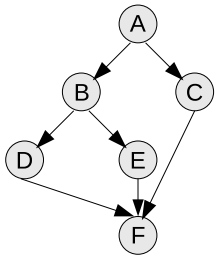

Fig. 1. A sample Task Graph. Task F must wait the termination of tasks C, D and E before starting.

The mapping process can be seen as a relation that associates t_i with the processing element m_j, $B : T \rightarrow M$, while the scheduling is a relation that associates each task t_i with a start time, $S : T \rightarrow \mathbb{N}$. A schedule is considered feasible when: *i)* task t_i is not started before all its immediate predecessors have ended and *ii)* the resource constraints are satisfied. There is a tight connections among scheduling and mapping: the scheduled tasks cannot exceed the available processing elements and the logic area on the FPGA, while the mapping should support the scheduling, trying to expose the maximum parallelism among the tasks.

Our approach imposes as few limitations as possible. In our formulation, the tasks can run on a single resource, on a subset of the resources or, potentially, on all the units available. For the FPGA, area constraints are supported, meaning that only a limited number of tasks can be allocated on the reconfigurable device. It is also possible to express the constraint to run a task on a specific resource instance. At the end of the execution, our algorithm provides a completely scheduled task graph with the allocation of each task to specific resource instances.

III. Ant Colony Optimization

The ACO methodology was originally introduced by Dorigo et *al.* in the form of the Ant System [4]. It was inspired by the observation of the behavior of ants when trying to reach a food source. A single ant essentially moves at random, but multiple ants can cooperatively search for the shortest path from their colony to the food. In fact, when ants move, they deposit a chemical substance, called pheromone, that is used for indirect communication. Ants are *motivated* to follow these trails: with high probability, random moving ants will go through the same path of previous ants. When they follow these trails, they deposit even more pheromones and reinforce them. At some point, some ants will reach the food and go back to their colony. Other ants may find other ways to the food. However, in the same amount of time, more ants will be able to go back and forth to the food following the shortest path rather than moving along longer routes. Thus, on that path, the concentration of the pheromone will grow faster, attracting more and more ants. The pheromone also has an evaporation factor. Therefore, as time goes by, pheromone on less reinforced routes will slowly disappear and, at some point, the only path with a strong trail will be the shortest one, making almost certain for all the ants to move along it.

ACO takes inspiration from this behaviour, launching several agents that explore the search space trying to find valuable solutions which, in turn, get reinforced. It adds, however, aspects like greediness (it knows the environment and thus can drive the ants to explore around interesting points of the search space) and memory of the produced paths to generate only feasible solutions. One of the first problems to which ACO has been successfully applied is the TSP [4], for which it gives competitive results when compared with traditional methods.

The Ant System for the TSP normally follows these steps:

1) Initially associate each arc with a pheromone trail τ_{ij}.
2) Put *m* ants on an initial city.
3) Each ant constructs its tour, executing a probability choice at each step from the set of allowed cities and memorizing the visited cities. The probability of going from city i to j is calculated as follows:

$$p_{ij} = \frac{[\tau_{ij}]^{\alpha} * [\eta_{ij}]^{\beta}}{\sum_{l \in N_i} [\tau_{il}]^{\alpha} * [\eta_{il}]^{\beta}}$$

   where N_i is the set of admissible choices for the ant at node i, η is a local heuristic that influences the choice of the ant on the next arc to explore starting from the current node, τ is the pheromone trail, while α and β controls the weight of the local heuristic and the global heuristic (pheromones).
4) The quality of the result is evaluated.
5) The pheromone trails are updated, firstly evaporating the trails on all the arcs and then incrementing them of a factor proportional to the goodness of the result found.
6) If goal conditions are not met, go to step 2.

The pheromones update formula is as follows:

$$\tau_{ij} = (1 - \rho) * \tau_{ij} + \sum_{l=l}^{m} \Delta\tau_{ij}^{(l)}$$

where $0 < \rho < 1$ is the evaporation rate and the deltas are calculated as $\Delta\tau_{ij}^{(l)} = Q/L$, if the arc ij was in the solution, 0 if not, with Q as pheromone delivery rate and L representing the cost of the result. Evaporation removes after each iteration some pheromones on all the arcs, allowing the ants to "forget" nodes explored and never revisited in order to not converge early to local minimums.

Goals can be a maximum number of generations, a fixed number of iterations for which the best results does not get better or convergence to a sub-optimum value by all the ants. The method originally proposed by Dorigo for pheromone update envisages a proportional update by all the ants launched in each iteration, at the end of the iteration itself. Several other different ways to update the pheromones have been proposed. Among them, we cite the *elitist approach*, in which only the best results of the current iteration are reinforced along with the overall best result, the *Max-Min* approach [8], that introduces adapting boundaries to the maximum and minimum values of pheromone trails to address possible premature convergence, and the *Ant Colony System* [9], which proposes an on-line mechanism to update the pheromone trails after every choice of each ant.

IV. Related Work

Mapping and scheduling are critical problems in the field of embedded system design that have been both solved with deterministic and non deterministic methods. Several ILP formulations have appeared for both the scheduling [10] and the mapping problems [11]. Nevertheless, these two problems are *NP-complete* and the optimal solution is computationally intractable for all but small instances. Thus, generally, optimization techniques, based on heuristic and evolutionary algorithms, that allow to reach suboptimal solutions in acceptable times are used. In particular, for the *Resource Constrained Scheduling Problem* (RCSP), the most common methods adopt a list based approach [12], which uses a priority list to determine the tasks to process first. With this approach, the quality of the results largely depends on the priority function used. Several techniques to obtain good priority lists have been proposed. A widely adopted priority function is the mobility, which assures that the most critical tasks are scheduled first, while the ones with the highest mobility can be deferred without increasing the overall execution time, thanks to their higher flexibility. Many heuristic search methods have been applied to obtain good priority functions. Among them, graph-theoretic and computational geometry approaches [13], SA, TS [14] and also GAs [15] have been the most successfull.

Several formulations have also been proposed for the *mapping* problem, defined as the problem to find the best hardware/software system partitioning in terms of assignment of the tasks to reconfigurable devices as hardware components or to general purpose processors as software threads. Deterministic and non deterministic methods have been applied to this problem too. Banerjee [16] et *al.* discussed an ILP formulation along with an heuristic method that takes care of the placement of the hardware tasks on the reconfigurable logic. A good number of overviews and comparisons of iterative search methods to solve this problem have been proposed [2]. Among them, the most popular are considered GA [17], TS and SA [18].

After the first formulation for the TSP, other ACO methods have been formulated to solve many traditional NP-hard problems. ACO, applied to the scheduling problem, has been studied by researchers of both the evolutionary computation and of the embedded system design fields. In particular, Merkle [19] et *al.* devised several solutions to adapt the Ant System techniques to the RCSP, introducing features like summation evaluation of pheromones trails, parameters with changing values from iteration to iteration, elitist solution discarding, local optimization strategies and bidirectional planning. Wang [7] et *al.* applied, instead, the Max-Min Ant System technique to both the resource and time constrained instruction scheduling problem. Wang [5] et *al.* also evaluated ACO in relation to the mapping problem, comparing its performance to SA and integrating the two techniques to improve the final solution.

Our work takes these studies as its basis, but introduces the ACO technique to solve both the scheduling and the mapping problem at the same time. With respect to Merkle's work [19], our problem is multi-modal. The standard RCSP formulation assumes that the type of resources requested by each project has been already fixed, while in our formulation, each task can, a priori, run on any of the compatible resources. Wang, instead, deals with scheduling and mapping separately, and does not combine the problems. In the mapping formulation, proposed in [5] he considers reconfigurable area limitations, like us, but extends the task graph representation in order to generate a data structure to allow the exploration by the ants. We do not extend it, since we allow each ant to perform a local mapping choice when required. In the resource constrained formulation for the instruction scheduling proposed in [7], Wang uses the ACO algorithm as an heuristic method to obtain the priority list for the list scheduler, while in our algorithm the ants directly construct the solutions. Furthermore, these works are centered around small problem instances, with no more than 350 nodes. In our formulation, each ant constructs step by step its solution, allowing the algorithm to explore around the good ones rather than searching solutions on a data structure that already contains all the possible permutations. To the best of our knowledge, this is the first formulation that tries to tackle both the scheduling and the mapping problem at the same time obtaining, at the end of each ant tour, a completely mapped and scheduled solution. Performing mapping and scheduling at the same time potentially allows to obtain much better results than performing a search only on one dimension, or first in one dimension and then in the other: although the search space becomes bigger, it permits to evaluate combinations that, using two different passes, without any feedback, can instead be cut early at the beginning of the search. Fixing the mapping reduces the search space for the scheduling. Viceversa, fixing a priority for the scheduling, makes some good mappings less effective.

V. Proposed algorithm

In this section we present the proposed ACO based algorithm. We show how it works starting with an example, and then we give some more details on the decision and pheromone update policies, on the complexity of the algorithm and on the optimizations implemented to obtain faster convergence to good solutions.

A. Example

We present the proposed algorithm with an example. Consider the task graph introduced in Fig. 1. In our formulation, each of the six tasks of the task graph can potentially be mapped on all the available resources. Each task is annotated with an estimation of its performance on each resource type. The estimation can be obtained with simulation, static or dynamic profiling or direct execution on the target processing element. Table I shows some possible annotations of a case in which each task can run on a DSP, a GPP and FPGA. We use simple numbers to allow a better understanding of the algorithm. Performance is expressed in clock cycles, FPGA area in terms of configurable logic bocks

Task	DSP Time	GPP Time	FPGA Time	FPGA Area
A	2	4	1	3
B	1	3	2	4
C	4	3	2	4
D	2	5	3	5
E	5	3	7	3
F	4	5	3	2

TABLE I
DISTRIBUTION OF THE EXECUTION TIMES AND AREA OCCUPATION ON THE DIFFERENT RESOURCES FOR THE EXAMPLE TASK GRAPH. THE TOTAL AVAILABLE AREA ON THE FPGA FOR THIS EXAMPLE IS 10.

For the FPGA, we also take in account the logic area occupied by the hardware implementation of each task. Even this value can be obtained with appropriate metrics. The reconfigurable logic has a different behaviour with respect to the other processing elements. In fact, it can allocate as many tasks as allowed by the available space and can launch them all in parallel if necessary. We consider only static allocation of tasks on the FPGA: when mapped, they cannot be removed, thus our model does not allow partial dynamic reconfiguration.

The *actions* of the ants, similarly to [19] are performed inside a Serial schedule Generation Scheme (SGS). This scheme constructs a complete schedule in N steps, where N is the number of tasks, selecting at each step a task with fixed resource requirements following a greedy approach. Using SGS, feasible solutions are always obtained, and it is known that, with appropriate choices, it can always reach the optimal solution for the RCSP [20]. The precedence constraints induced with the directed task graph are managed generating at each step a list of candidate nodes. From this candidates list, the SGS selects the effective node to schedule and map in the current step through a heuristic, which can for example be a priority list, similarly to a list scheduler. Like [19] we consider each step of the SGS as a *decision* point for the ants.

Returning to our example, an ant starts crawling to task A. A standard SGS would just select the task, check the pre-assigned resource type and schedule the task as soon as a resource of that type is available (i.e., all previously selected tasks running on the same unit have ended). For task A, it would just allocate it at time 0 on its target resource. In our formulation, instead, at this point an ant already has three different possibilities as step *i*) in Fig. 2 shows: it can select A on the DSP, on the GPP or on the FPGA. Each decision has its probability, and a probabilistic choice is made. Suppose the ant selects the instance *task A on FPGA*. Thus, A is mapped on the FPGA and the SGS scheduler places it at *time 0* on that resource. Consequentely B and C become available for mapping and scheduling. Both B and C can run on the three resources, so the ant now has 6 possible choices, that represent the chance of scheduling a node on a unit in the current scheduling step. Suppose that, in this step, *task B* is chosen, again, *on the FPGA*. Thus, B is scheduled just after A, at *time 1*, on the

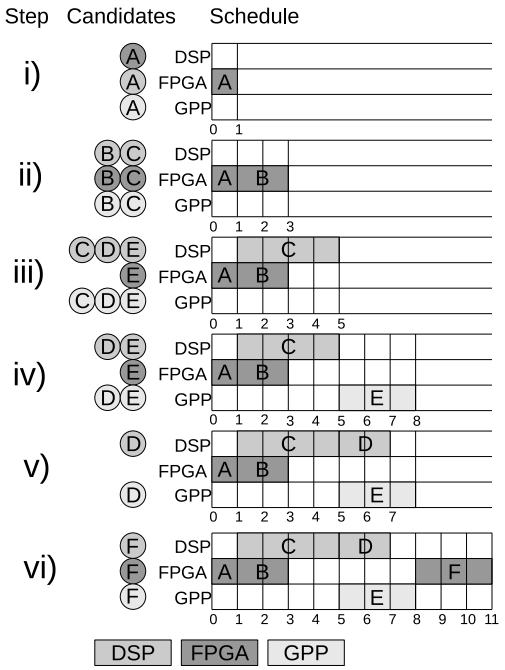

Fig. 2. The six steps performed by an ant while scheduling our example task graph

FPGA, as shown in step *ii*) of Fig. 2. Now also D and E become admissible candidates along with C for the next SGS step. However, the total area on the FPGA in our example is 10. Seven units of space have already been allocated, thus the possible choices for the ant at this stage are: C, D, E on DSP and GPP, but only E on FPGA. So the ant has to choose among 7 possibilities instead of 9, as reported in step *iii)* of Fig. 2. Probabilities are dynamically generated at each step, as proposed in the original formulation of the Ant System for the TSP. But, in this case, all the solutions obtained at the end of the ant tour are feasible. The ant makes again its probabilistic choice, and selects *task C on the DSP*. C can start at time 1, completely in parallel with task B, since the only needed precondition to obtain a feasible schedule is to start B after task A has ended. The ant could have chosen task D or E, but the SGS would have scheduled them after the end of task C to respect the dependencies. Furthermore, if the same unit is chosen for tasks that can potentially run in parallel, the SGS would take care of the resource requirements, scheduling the new task as soon as the target unit is available. The next step (*iv*) for the ant is to schedule and map task D or E. Since only one of the three predecessors of F has been scheduled at the current step, F is not added yet to the candidates list. *E* is chosen, mapped and scheduled at *time 5 on the GPP*. Then, only D remains in the candidates list with the probabilities to run DSP and GPP. Remember that the FPGA has already 7 space units allocated so far, thus D cannot run on the FPGA in the current schedule. *D* is selected *on the DSP* and scheduled at *time 5*, so finally also *F* can be scheduled and mapped. F has an area occupation of 2, so it can potentially run also *on the FPGA*. It is then scheduled on it at *time 8*, just after its last predecessor, E, ends. The complete schedule generated by this ant is reported in step *vi)* of Fig. 2.

B. Probability decisions and pheromone update

Fig. 3 shows the decision graph, only for the scheduling part, for our example. It is easy to see that, only for the scheduling decisions, the size of this structure is twice the size of our initial task graph. Potentially, considering a task graph with N nodes, with a first task that spawns all the possible children (*fork*), and a final task that acts as a *join* node, it can grow up to $N(N-2)$ vertices. If, for each task, M different target units are possible, it is necessary to explore a graph of $N(N-2)M$ vertices. Our ACO formulation explores only promising parts of this graph, reinforcing the good triplets *<task, resource, step of selection by the SGS>* chosen in the best solutions.

Following these concepts, we initially devised a mechanism that unifies both mapping and scheduling in a single decision formula by the ants. At each step i, we calculate the probability to select task t on the unit m as:

$$p_{itm} = \frac{[\tau_{itm}]^{\alpha} * [\eta_{itm}]^{\beta}}{\sum_{l \in N_t m} [\tau_{il}]^{\alpha} * [\eta_{il}]^{\beta}}$$

η is a local heuristic and it is calculated everytime a probability is generated. Different values can be used, like mobility, estimation of the critical path, latest finishing times of predecessors nodes, or the earliest starting time of a task (the maximum among the latest finishing time of its predecessors and the availability of the target resource), possibly adding the performance of the task on the target unit in order to characterize the mapping.

τ is the global heuristic, which corresponds to the *pheromones* of the ants and which is updated in each iteration of the algorithm, if the ants included that choice in their solutions. ACO stores these reinforcements in a *pheromone matrix*. This means that a value is saved for each triplet <task, target resource, scheduling step>, generating a structure as big

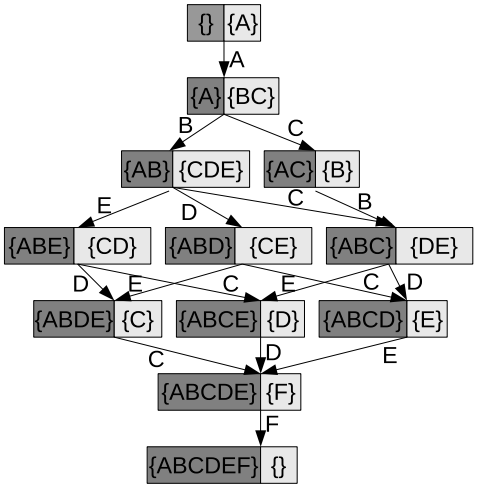

Fig. 3. The decision graph for the scheduling selection on which ants *crawl* while constructing a solution. Only part of it is explored by a each ant.

as the completely unrolled decision graph. In the multi-modal case, many of these locations can be considered equivalent: if task A and B run in parallel, and are mapped on different, free resources, changing the order in which these two nodes are considered by the scheduler does not change the final outcome. Thus, it can make the reinforcements less effective and the convergence to the optimum slower.

With these considerations in mind, we devised a two-stage decision process. At each scheduling step, 2 probabilities are calculated:

$$p_{it}^s = \frac{[\tau_{it}^s]^{\alpha^s} * [\eta_{it}^s]^{\beta^s}}{\sum_{l \in N_i} [\tau_{il}^s]^{\alpha^s} * [\eta_{il}^s]^{\beta^s}}$$

which each ant uses to determine the task t to schedule at step i, and:

$$p_{tm}^m = \frac{[\tau_{tm}^m]^{\alpha^m} * [\eta_{tm}^m]^{\beta^m}}{\sum_{l \in N_t} [\tau_{tl}^m]^{\alpha^m} * [\eta_{tl}^m]^{\beta^m}}$$

which is calculated after the task to schedule has been selected, and expresses the probability that the selected task t will be mapped on the resource m. If a task cannot be allocated on the FPGA for lack of available area, that mapping probability is not generated. The final result of the decision process is still the mapped task to schedule in the next scheduling step. The correlation among scheduling and mapping is maintained. However now the process is much more controllable. Scheduling is performed by computing the probabilities on a pheromone matrix, $\mathbb{T}^s$ of $N * N$ elements, representing the possibility to select a node in the current step. Mapping is based on another matrix, $\mathbb{T}^m$, of size NM which contains the reinforcements for mapping a task on a resource. Total dimensions are now $NN + NM$. For each stage of the decision process, a specific, more effective, local heuristic can be implemented. Moreover, different evaporation rates and parameters, to fix scheduling before mapping or viceversa, can be used.

We apply the pheromone update following an elitist approach: only the best solution of an iteration adds pheromone deltas to the mapping and scheduling choices, along with the current overall best solution. When an ant ends its tour, the task graph is completely mapped and scheduled, and its overall execution time is obtained. This information is then used to proportionally reinforce the elements chosen in the best solutions by the ants. The update formula, executed for the best solution of each iteration and for the current overall best solution is as follows:

$\tau_{it}^s = (1 - \rho^s) * \tau_{it}^s + \rho^s * \frac{1}{L}$ for the scheduling pheromones, and $\tau_{tm}^m = (1 - \rho^m) * \tau_{tm}^m + \rho^s * \frac{1}{L}$ for the mapping. L is the total length of the scheduled and mapped task graph.

C. Other Optimizations

With this two-stage mechanism, all the solutions devised by Merkle in [19] to favour the exploration and to speed up the convergence to good solutions for the RCSP can be applied. In particular, we adopt the summation evaluation for the pheromone trails (only for the scheduling), the possibility to forget the elitist best solution and the 2-opt local optimization.

Summation evaluation is a pheromone evaluation rule originally introduced for the Total Tardiness Problem [21] that takes into account the relative influence of pheromones values corresponding to earlier decision. It can help the scheduler since with the SGS scheme several permutations in the selection of the tasks to schedule are possible. Summation evaluation is determined using:

$$p_{it}^s = \frac{[\eta_{it}^s]^{\beta^s} * [\sum_{k=1}^{i} \gamma^{i-k} \tau_{kt}^s]^{\alpha^s}}{\sum_{l \in N_i} [\eta_{il}^s]^{\beta^s} * [\sum_{k=1}^{i} \gamma^{i-k} \tau_{kl}^s]^{\alpha^s}}$$

It can be combined with direct evaluation by simply replacing the τ_{it}^s of the direct evaluation formula with the new:

$$\tau_{it}'^s = c x_i \tau_{it}^s + (1 - c) y_i \sum_{k=1}^{i} \gamma^{i-k} \tau_{kt}^s$$

where $x_i \sum_{l \in N_i} \sum_{k=1}^{i} \gamma^{i-k} \tau_{kl}^s$ and $y = \sum_{l \in N_i} \tau_{il}^s$ are factors to adjust the relative influence of direct and summation evaluation. For $c = 1$ pure direct evaluation is obtained, while for $c = 0$ pure summation evaluation is used.

Changing the current best solution, with low probability, allows the algorithm not to converge too early. If the best solution has been stable for many generations it gets constantly reinforced by the elitist pheromone update mechanism. If this solution has no other interesting points in its neighborhood, the search will slowly concentrate around its region, since the evaporation rate will make the elements of this solution the most reinforced ones. The algorithm would thus converge to this local optimum cutting out the possibility to find better solutions in different regions.

Finally, local optimization strategies help the ants to perform faster searches in the neighborhood of good solutions [22]. We adopt a 2-opt strategy that, with different probabilities, can swap or the order in which two tasks get selected for the scheduling, or the mapping of a task. This strategy is applied to the best solution at the end of each iteration. When swapping the order in which tasks are selected, we verify if the new scheduling is feasible; when changing the mapping of a task, we recalculate the total execution time with the new resource assignments. These simple swaps are likely to be tested by the ants, so our optimization strategy simply makes the search around the best solution faster.

VI. Evaluation

In this section we evaluate our algorithm by applying it to several test cases. We begin presenting the experimental setup and the algorithms used for comparison, and we then discuss the results obtained.

A. Experimental Setup

To evaluate our ACO algorithm, we randomly generated several realistic task graphs using Task Graph For Free [23], with 100, 200, 250, 500, 750 and 1000 nodes (with ± 5 variance). For the 200 and 500 instances, we generated multiple solutions with different degrees of dependencies. An XML

architecture description file provides to our tool a target architecture composed of 4 heterogeneous processing elements: a DSP, an ARM GPP, a PowerPC GPP, and a FPGA. Each task was generated with different performance for each processing element (e.g., a performance of 1600 ± 1500 clock cycles for ARM) and an area of 5000 ± 2000 logic cells for execution on FPGA. FPGA total available area was configured to $100,000$ logic cells. We implemented two other standard search algorithms, SA and TS, which are widely used for task mapping, and we adapted them to deal with multiple resources and with the scheduling problem. Both TS and SA are adaptations from the *Neighborhood Search* (NS), a hill climbing algorithm. Unlike NS, 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 temperature drops below a certain threshold, the algorithm ends. Several cooling schedules can be used, among them the most common are the geometric schedule and the Lundy and Mees schedule. TS, instead, exploits a search history as a condition for the next moves. When generating new solutions, TS checks its short term memory to avoid searching the same neighborhood (*tabu status*). After some time, however, the tabu status will be released and these solutions will become eligible again. The use of tabu restrictions can however sometimes stop the search in promising areas, thus it is common to introduce *aspiration criteria*, that override the tabu state and permit the inclusion of good solutions in the allowed set. A common criterion is to not respect the tabu status if a move permits to obtain a solution better than the best so far.

In our formulations for SA and TS, each solution is composed of two elements: a map in which each task is associated with a target unit, and a priority list which is given to the SGS scheduler for the selection of the task to schedule. The generation of a neighbor coincides with the random change of the mapping of a task or the random swap of the priority among two tasks. The starting scheduling priority list is based on the topological order. Tasks are scheduled by the SGS verifying precedence and resource constraints and then scheduling priority, so feasible schedules are always obtained.

B. Experimental Results

We executed 30 runs on each task graphs produced by TGFF and averaged the results. The following parameters were used to conduct the simulations. For the ACO algorithm, $\alpha^s = \alpha^m = \beta^s = \beta^m = 1$, were used along with different evaporation rates for the scheduling ($\rho^s = 0.9$) and the mapping ($\rho^m = 0.98$). This choice is motivated by the fact that scheduling is a harder problem and a faster evaporation rate helps the ants to fix at least a local optimum sooner. As local heuristics, for the scheduling decision we adopted the mobility of the tasks (we perform ALAP and ASAP scheduling to obtain an average value without taking into consideration the resource compatibility of the tasks), while for the mapping decision we used the inverse of the sum of the maximum time between the target resource availability and the finishing time of the lastest predecessor of the task with the execution time of the task on the target resource. For the FPGA, however, we just take into account the finishing time of the latest predecessor of the task, since if the task can be mapped on it (i.e., there is space available), the resource is always available. 5 ants for a maximum of 2000 iterations were used. For the SA algorithm, we used a geometric cooling schedule, $T_{new} = \alpha T_{old}$, with $\alpha = 0.99$. Initial temperature T_{start} was set to 250, while T_{finish} was 0.25. For TS, we generated 10 neighbors for 1000 iterations with a tabu list composed of 10 sets of solutions each. The tabu list acts as a FIFO queue: a set of freshly generated neighbors, not present in previous sets, is inserted in each iteration. The older sets have a lower tabu degree so, if following the aspiration criterion a result is chosen from the tabu list, a choice in the sets coming from older iterations is preferred.

Table II shows, for each of the three search algorithms, the average best solutions obtained in mapping and scheduling the annotated tasks graphs, the standard deviation from the average best solutions (σ) and the execution time of the algorithm until convergence or reaching of the maximum number of allowed iterations. The solutions are calculated as the total number of clock cycles required to execute the mapped and scheduled task graphs. Average of the best solutions found by SA and TS are presented as percentage difference from the average best solutions found *by ACO*. Standard deviation is represented as percentage variation from the average best solutions found *by each search algorithm*. The values show that our ACO formulation, when compared with other hill climbing derived search algorithms that try to solve the combined mapping and scheduling problems, globally performs better, faster and appears to be more robust. With 100 nodes, SA almost reaches the quality of the results obtained with ACO, even if with a slightly higher variance and more than double execution time. We do not report values for task graphs smaller than 100 nodes, since with those sizes it is still possible to obtain good results with an exhaustive search and SA, with repeated small local modifications, can obtain results of the same quality of ACO. With task graph instances from 100 to 250 nodes, SA generally performs better than TS but, with a much higher standard deviation, appears to be less robust. When the task graphs grow to 500 nodes and more, however, SA is no more able to manage the search in the combined scheduling and mapping space. The temperature starts to diverge, and the search concludes very soon, as the execution times of the algorithm show. As a consequence, the quality of the best solutions found gets much worse with respect to both ACO and TS. Generally, TS is more robust than SA, but cannot reach the same robustness of ACO. ACO, thanks to the use of two heuristics, local and global, is more easily driven to good results from the beginning of the search. Since TS does not implement a reward mechanism for good solutions, it requires higher execution time (almost double) to find its best solutions but, thanks to the tabu list, it can limit its search only to promising neighborhoods. On our test cases the ACO formulation for mapping and scheduling can find solutions that, in average, are respectively 64% and 55% better with

Tasks	ACO			SA			TS		
	Av. Best	σ	Time (s)	diff.	σ	Time (s)	diff.	σ	Time (s)
100	51.403	±1,54%	14,45	+1,31%	± 6,67%	33,43	+64,33%	±8,10%	40,38
200-a	107.230	±1,22%	32,92	+36,27%	±32,17%	109,07	+52,55%	±5,85%	112,73
200-b	60.080	±1,70%	57,01	+23,19%	±34,54%	122,72	+54,19%	±4,96%	124,70
200-c	65.876	±2,47%	51,81	+59,99%	±28,86%	48,56	+56,57%	±5,89%	112,40
250	142.440	±0,84%	49,93	+38,92%	±28,09%	149,78	+57,03%	±5,93%	170,75
500-a	300.938	±1,14%	269,19	+82,30%	±16,78%	119,44	+70,36%	±8,07%	501,86
500-b	163.694	±1,41%	207,91	+84,86%	±19,30%	225,08	+62,92%	±6,78%	494,63
500-c	181.315	±1,60%	253,32	+103,38%	± 9,99%	72,98	+66,72%	±7,36%	446,01
750	445.936	±1,65%	520,20	+102,17%	± 9,38%	220,02	+80,98%	±6,24%	897,79
1000	617.912	±2,63%	820,39	+105,33%	± 4,89%	305,29	+84,95%	±6,26%	1467,35

TABLE II
COMPARISON AMONG THE SEARCH METHODS APPLIED TO SOLVE THE COMBINED SCHEDULING AND MAPPING PROBLEM. THE AVERAGE BEST SOLUTIONS FOUND BY SA AND TS ARE NORMALIZED TO THE AVERAGE BEST RESULTS FOUND BY THE ACO ALGORITHM.

respect to SA and TS.

VII. CONCLUSIONS

This paper presented the details of a novel formulation for an ACO search algorithm targeted to solve the combined mapping and resource constrained scheduling problem. The algorithm also supports reconfigurable logic and deals with area limitations when allocating tasks on FPGA. Compared to other ACO formulations, this is the first approach that tries to solve, at the same time, the two NP-hard problems together. The implementation of a two-stage decision mechanism allows a correlated decision on the scheduling and the mapping by the same ant, simplifying the pheromone data structures but not limiting the search space. We demonstrate that our formulation, compared to SA and TS search algorithms, adapted to deal with both mapping and scheduling at the same time, performs better, faster and more robustly. Overall, it can obtain solutions, respectively, 64% and 55% better than these two hill climbing derived search algorithms. Future works will focus on extending the algorithm to support the concept of partial dynamic reconfigurability, which allows the reuse of the FPGA resources but requires to take into account some new overheads and some new constraints in the placing of the tasks on the programmable logic.

REFERENCES

[1] D. Bernstein, M. Rodeh, and I. Gertner, "On the complexity of scheduling problems for parallel/pipelined machines," *IEEE Trans. Comput.*, vol. 38, no. 9, pp. 1308–1313, 1989.

[2] T. Wiangtong, P. Cheung, and W. Luk, "Comparing three heuristic search methods for functional partitioning in hardwaresoftware codesign," *Design Automation for Embedded Systems*, vol. 6, no. 4, pp. 425–449, July 2002.

[3] T. Lei and S. Kumar, "A two-step genetic algorithm for mapping task graphs to a network on chip architecture," in *Digital System Design, 2003. Proceedings. Euromicro Symposium on*, Sept. 2003, pp. 180–187.

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

[5] 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.

[6] H. S. Daniel Merkle, Martin Middendorf, "Ant colony optimization for resource-constrained project scheduling," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, D. Whitley, D. Goldberg, E. Cantu-Paz, L. Spector, I. Parmee, and H.-G. Beyer, Eds. Las Vegas, Nevada, USA: Morgan Kaufmann, 10-12 2000, pp. 893–900.

[7] 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.

[8] T. Stutzle and H. Hoos, "Max min ant system," *Journal of Future Generation Computer Systems*, vol. 16, pp. 889–914, 2000.

[9] M. Dorigo and L. M. Gambardella, "Ant colony system: A cooperative learning approach to the traveling salesman problem," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 53–66, April 1997.

[10] K. Wilken, J. Liu, and M. Heffernan, "Optimal instruction scheduling using integer programming," in *PLDI '00: Proceedings of the ACM SIGPLAN 2000 conference on Programming language design and implementation*. New York, NY, USA: ACM, 2000, pp. 121–133.

[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. L. Adam, K. M. Chandy, and J. R. Dickson, "A comparison of list schedules for parallel processing systems," *Commun. ACM*, vol. 17, no. 12, pp. 685–690, 1974.

[13] A. Aleta, J. Codina, J. Sanchez, and A. Gonzalez, "Graphpartitioning based instruction scheduling for clustered processors," 2001.

[14] S. J. Beaty, "Genetic algorithms versus tabu search for instruction scheduling," in *International Conference on Neural Network and Genetic Algorithms*. Springer, February 1993, pp. 496–501.

[15] M. Grajcar, "Genetic list scheduling algorithm for scheduling and allocation on a loosely coupled heterogeneous multiprocessor system," in *DAC '99: Proceedings of the 36th ACM/IEEE conference on Design automation*. New York, NY, USA: ACM, 1999, pp. 280–285.

[16] 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.

[17] J. I. Hidalgo and J. Lanchares, "Functional partitioning for hardware-software codesign using genetic algorithms," in *23rd Euromicro Conference*, 1997, pp. 631–638.

[18] 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, pp. 5–32, 1997.

[19] 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.

[20] S. Hartmann and R. Kolisch, "Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem," *European Journal of Operational Research*, vol. 127, no. 2, pp. 394–407, December 2000.

[21] D. Merkle and M. Middendorf, "An ant algorithm with a new pheromone evaluation rule for total tardiness problems," in *Real-World Applications of Evolutionary Computing, EvoWorkshops 2000: EvoIASP, EvoSCONDI, EvoTel, EvoSTIM, EvoROB, and EvoFlight*. London, UK: Springer-Verlag, 2000, pp. 287–296.

[22] M. Dorigo and G. Di Caro, "The ant colony optimization meta-heuristic," in *New Ideas in Optimization*, D. Corne, M. Dorigo, and F. Glover, Eds. London: McGraw-Hill, 1999, pp. 11–32.

[23] R. P. Dick, D. L. Rhodes, and W. Wolf, "TGFF: task graphs for free," in *Hardware/Software Codesign, 1998. (CODES/CASHE '98) Proceedings of the Sixth International Workshop on*, Seattle, WA, Mar. 1998, pp. 97–101.