An ACO-based, Reconfiguration Aware Mapper and Scheduler

Riccardo Cattaneo, Christian Pilato, Gianluca Durelli, Alessandro A. Nacci, Marco D. Santambrogio, Donatella Sciuto

POLITECNICO DI MILANO — NECST laboratory — FASTER

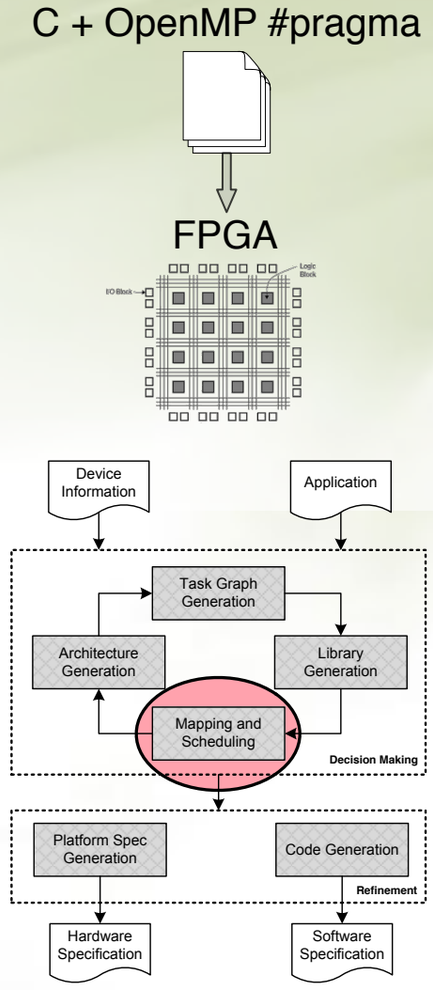
Context Defintion: the FASTER project at a glance

Faster is a EU research project aimed at facilitating the use of reconfigurable technology by providing a **complete methodology** that enables designers to **easily implement** and **verify applications** on platforms with general-purpose processors and acceleration modules implemented in the latest **reconfigurable technology**.

Users specify both the **application** (written in C with OpenMP #pragmas) and the **architectural template** of the final solution to be implemented on FPGA. The toolchain **automatically devises** a very **good solution** in terms of architecture specialization, application partitioning, mapping, scheduling and runtime management.

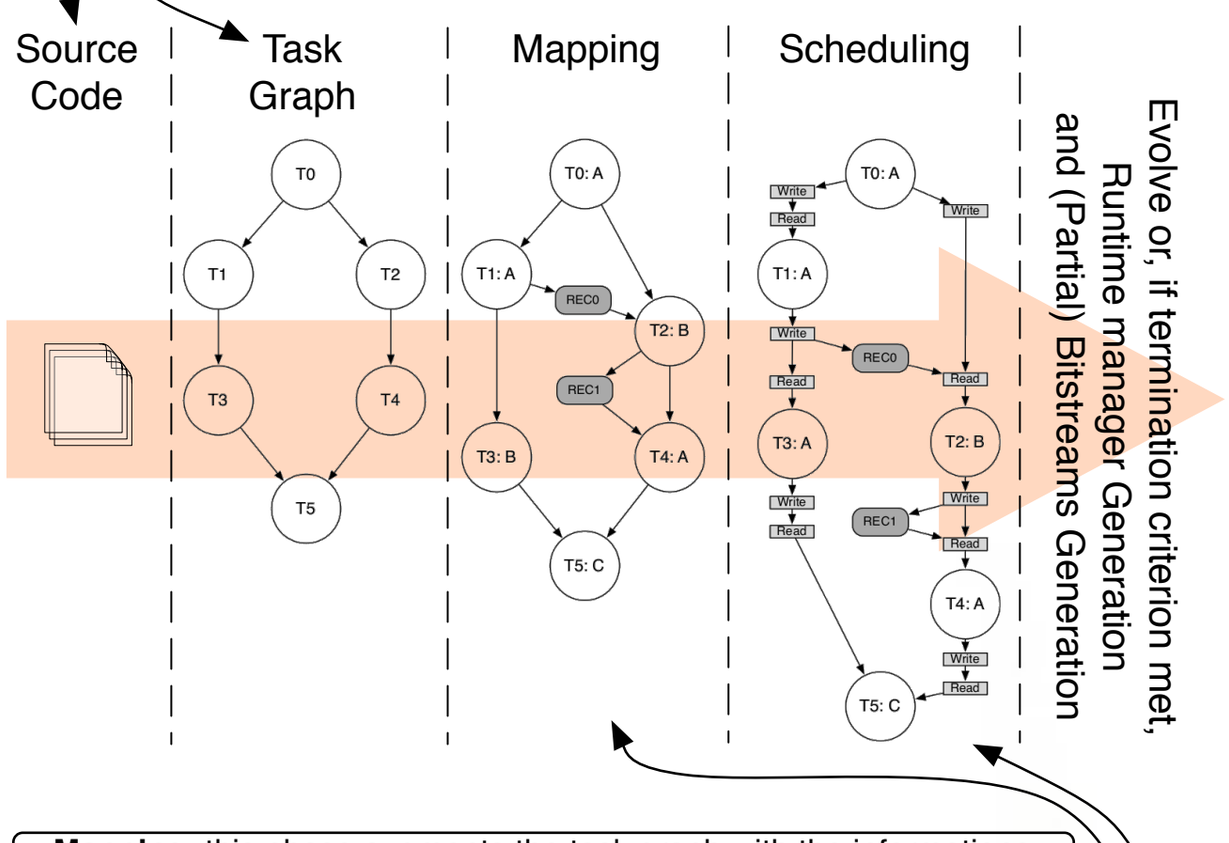
While most of the parameters (even the architectural template) can be guessed automatically by the toolchain, the **designer** can interact with the tools at each step of the exploration. This allows her to **trade off simplicity** of porting the application to hardware **with** potential **performance improvements**

This work revolves around the mapping and scheduling phase of FASTER.



Reconfiguration Aware Mapping and Scheduling

First two phases: **the source code**, which is written in C + OpenMP #pragmas, is **analyzed**. The functions that have been adequately marked are assigned a node in the task graph along with the producer/consumer relationship. The **architectural template** and the set of **tasks' implementations** (i.e.: ways to execute a tasks on a specific class of processors) is fed into the **mapper** as well. The **implementations** are characterized by **estimates** on the **number of resources** required to execute them (LUTs, BRAMs, etc…) as well as **estimates on** the expected **performance**.



Mapping: this phase augments the task graph with the informations incoming from the exploration phase, in which an optimization algorithm finds the **"best" mapping**. In this work, we devise this mapping by means of an **Ant Colony Optimization**-based algorithm where we **minimize the overall execution time** of the resulting solution, while guaranteeing that the **final solution is admissible** in terms of total resource consumption. In order to take into account how reconfiguration impacts on the system, **reconfiguration tasks are introduced** between tasks mapped onto the same reconfigurable regions and their execution times are estimated.

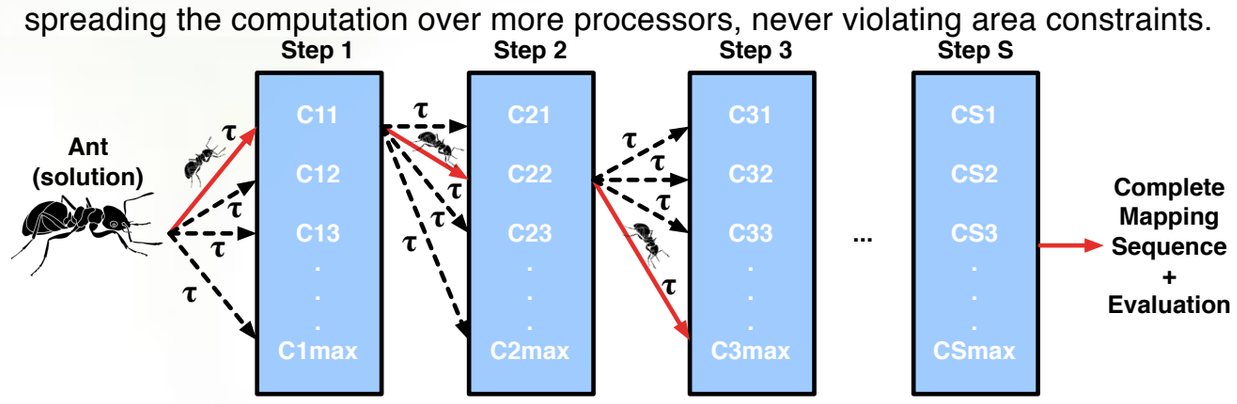
Scheduling: this phase introduces communication tasks in the taskgraph and assigns a start and end times to each of them in the augmented taskgraph. In this work, we employ a **priority list based scheduler** to keep exploration time low (scheduling phase easily becomes the bottleneck in the execution of the exploration loop). After assigning a proper start and end time to each task, the **total makespan** is computed and the **mapping is scored** proportionally to how fast it allows the application to complete. If the termination criteria of the exploration algorithm is not met yet, the performance metrics associated to this solution are used to let the system evolve and improve the solutions of the next generation.

ACO: methodology

Ant Colony Optimization (**ACO**) is a **metaheuristic optimization** algorithm. The **solution** (in the sense of the objective function) is **incrementally built** as a **sequence of choices**. At each step, **each choice** pertaining to the given step is assigned a "**goodness**" **value**. The higher the value, the higher the chances that the choice is made at that step (the **choice** is always **stochastic - roulette wheel selection**). Two functions are involved in the evaluation process, balancing the design space exploration between a **local** and a **global search**: τ and η.

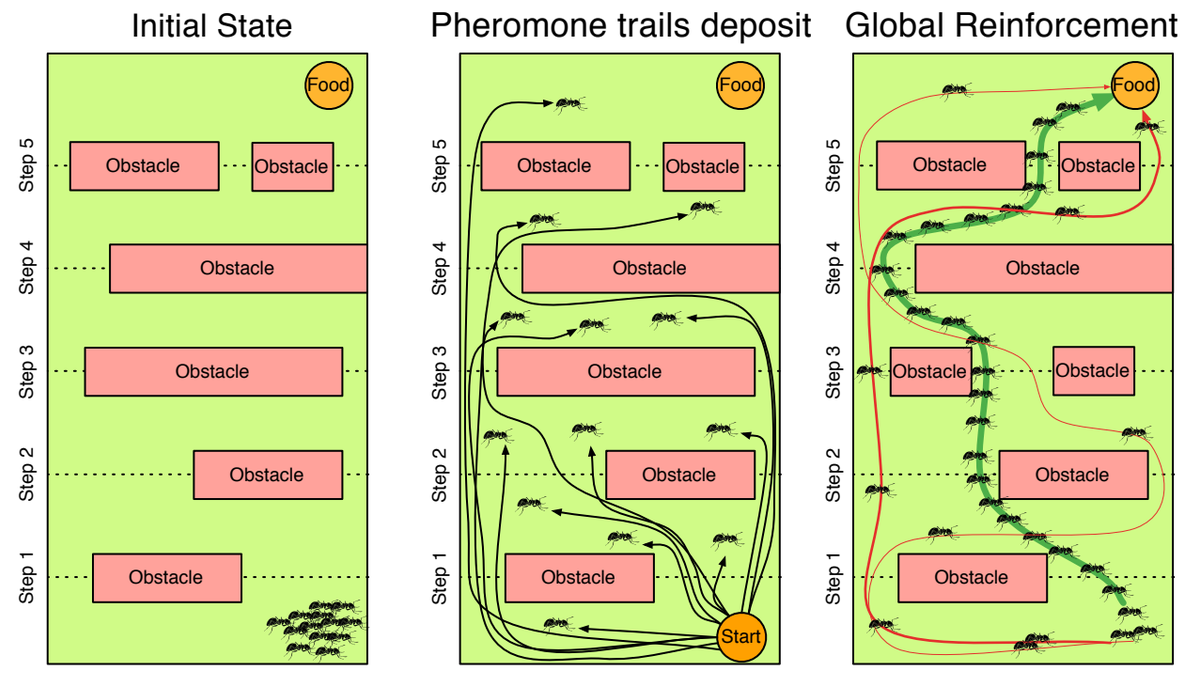
Local Search: τ(choice)

This function considers **local state only** when making a choice; implements a reasonable heuristic for a **local (greedy) search**. We employ two local search functions to select the task, before, and the implementation X processor pair, after. The heuristic tries to reduce the overall average mobility on a processor while spreading the computation over more processors, never violating area constraints.



Global Search: η(choice)

This function considers **global state only** when making a choice; implements the metaphor of **pheromones** as a matrix of weights. Pheromones are deposited on trails during exploration; the best solutions (ant's trails) are reinforced by other ants, the others gradually evaporate and fade out, thus leaving behind only the best track.

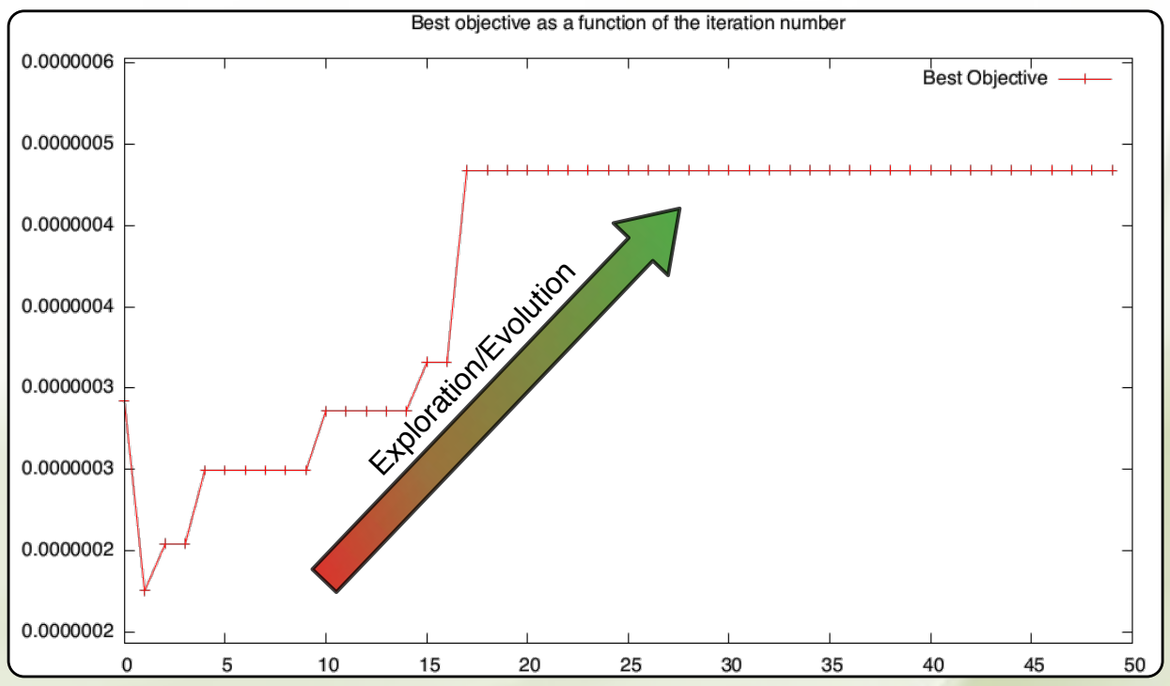


Rationale of ACO for Mapping and Scheduling

If a specific mapping at a given step results in a good overall execution time more frequently than not, try to build mappings which have that choice at that step more frequently than not. Tend to reinforce those choices while the system evolve by means of the pheromone metaphor.

Preliminary Results and Future Work

We report an example run of one of our experiments, carried out with a synthetic application with **100 tasks** randomly connected to form a DAG, each with multiple hardware ad software implementations. The application is **automatically mapped** onto a reconfigurable architecture with up to **30 reconfigurable regions**, two soft processors. The final number of reconfigurable regions actually used is 27 (the template is successfully specialized). The overall **area constraint is respected**. The exploration algorithm runs is **less than 2 hours** on a commodity Intel Quad Core i7, 8 GB of RAM.



In the continuation of this work we are aiming at the introduction of exact methods for solving the mapping and scheduling problem (for more performant mappings but slower exploration times), a better integration with the rest of the toolchain and more extensive evaluation of the overall approach, in particular against real life applications.