POLITECNICO DI MILANO

2nd Workshop on Mapping of Applications to MPSoCs

Mapping and Scheduling of Parallel C Applications with Ant Colony Optimization onto Heterogeneous Reconfigurable MPSoCs

Fabrizio Ferrandi
Dipartimento di Elettronica ed Informazione
Politecnico di Milano
ferrandi@elet.polimi.it

Outline

- Introduction and Motivation
- Preliminaries and Problem Definition
- Proposed Methodology
- Experimental Results
- Conclusions and Future Work

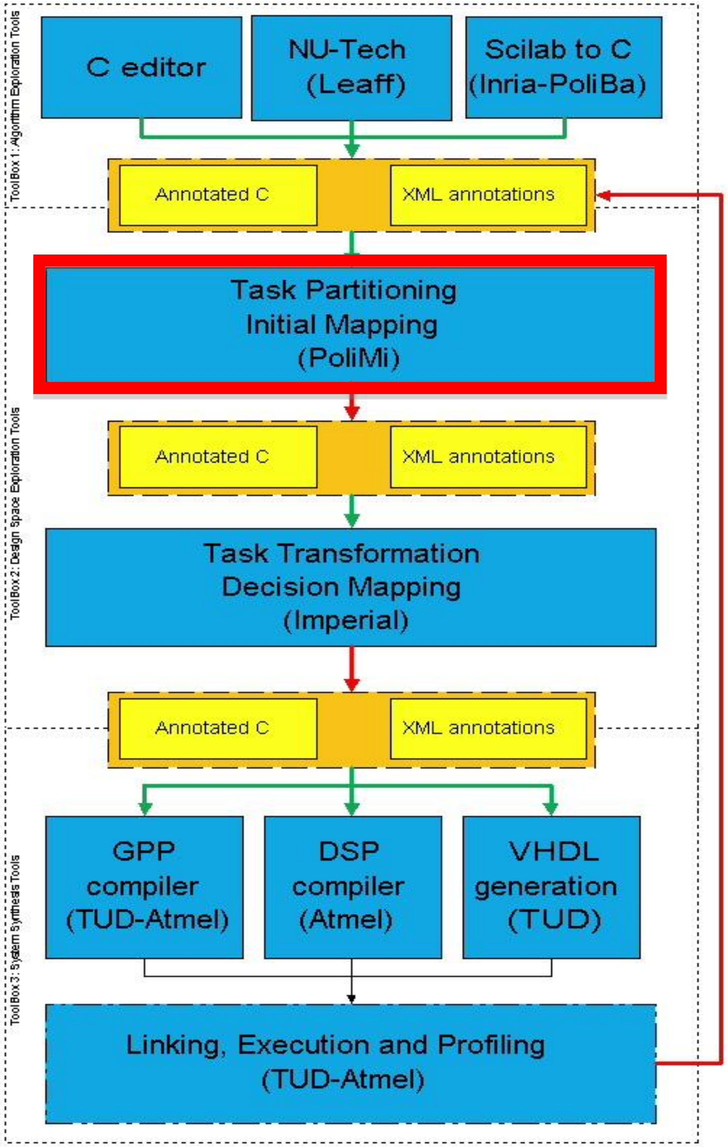
Introduction

- The design of application for Multi-Processor System on Chip requires to:
  - Partition the application (*task partitioning)*
  - Assign the tasks to the processing elements (*mapping)*
  - Determine the order of execution of the tasks (*scheduling)*
- Scheduling and mapping are *NP-complete* problems
- Additional problems due to heterogeneous components and design constraints
  - Possibility to generate unfeasible solutions

hArtes Project

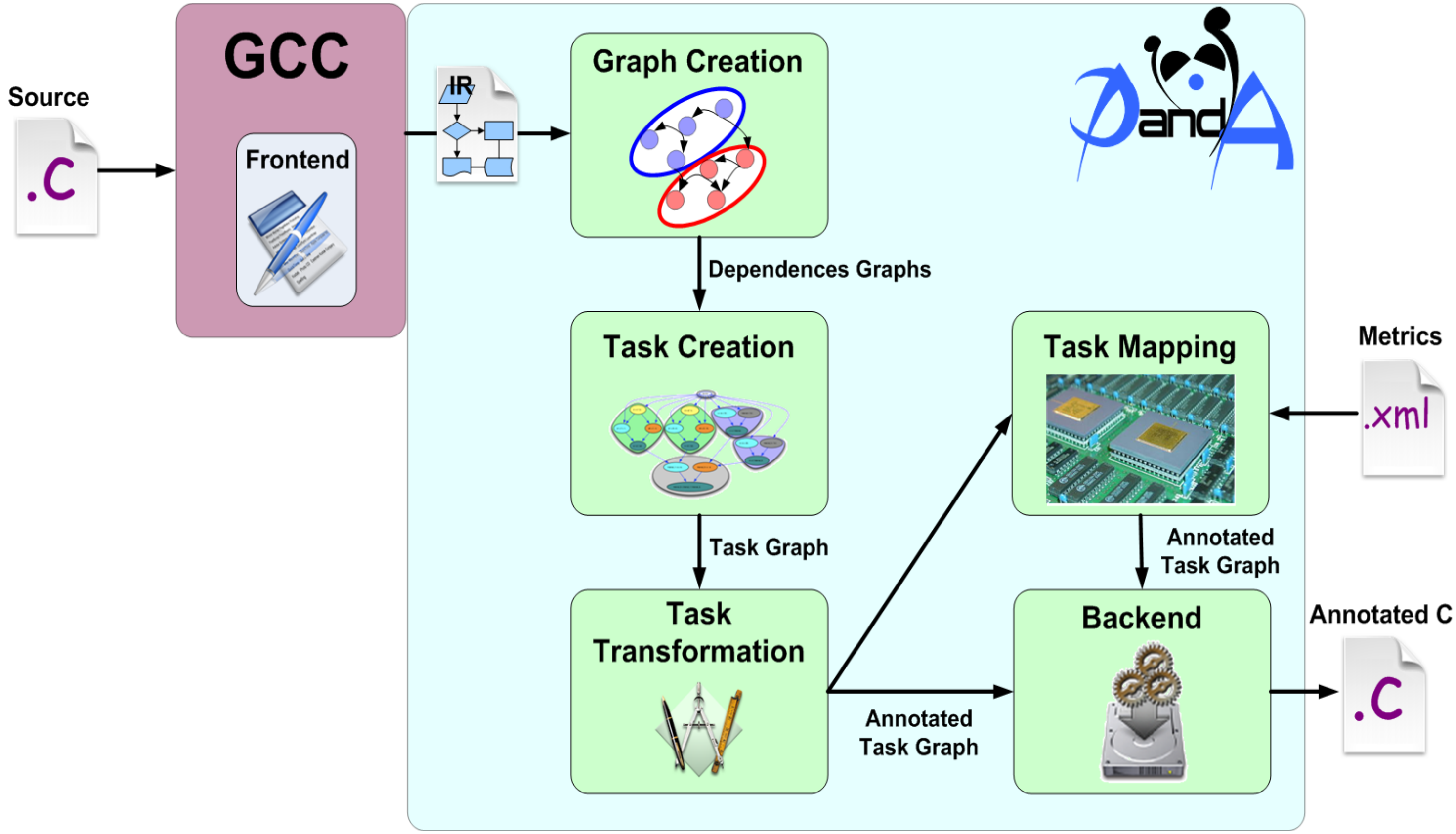
- Integrated European Project in FP6 (2006-2009)
- New holistic (end-to-end) approach for complex real-time embedded system design:
  - support for different formats in algorithm description;
  - a framework for design space exploration, which aims to automate design partitioning, task transformation, and metric evaluation for all the components;
  - a system synthesis tool producing near-optimal implementations that best exploits the capability of each type of processing element.
- We rely on C-to-C transformations and pragma insertion to represent partitioning and mapping
  - Each task is represented by a function

hArtes Overview



- We have in charge the **task partitioning** of the initial specification and an initial guess of **mapping**
- In this presentation we focus on the mapping phase
  - Analyze a partitioned application
  - Identify the candidate processing element for the execution of the tasks
  - Generate the related pragmas

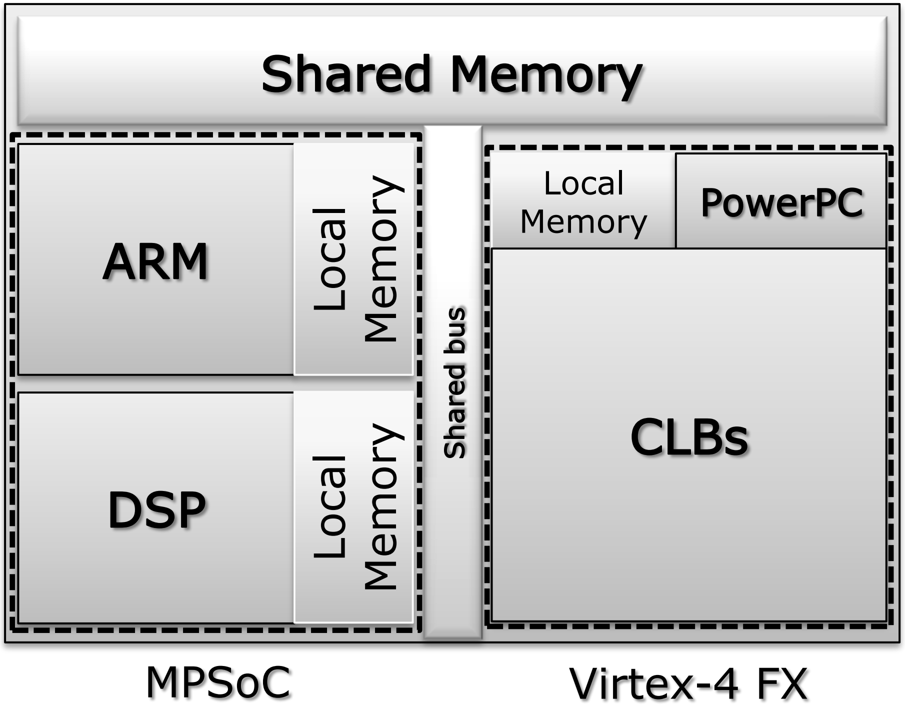
PandA Framework for task partitioning and mapping



Note: In this presentation, we focus on Task Mapping

Target Architecture

- Generic architectural template composed of processing and communication elements. A valid test case is the following one:



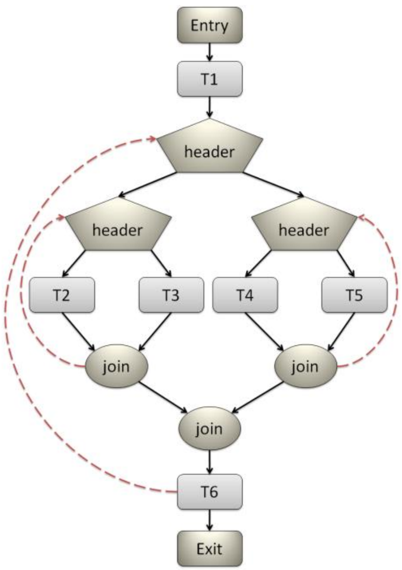
Renewable (e.g., local memories, bandwidth) and *non-renewable* resources (e.g., hw area) associated with all the components

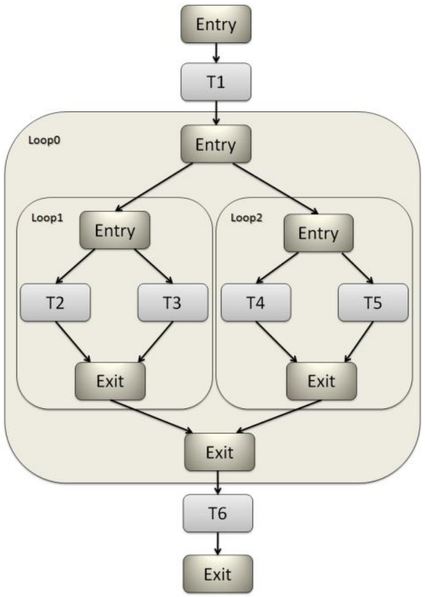
Application Model

- A **Task Graph** is a graph G=(T,E) which nodes represent group of instructions and edges represent dependences.
- Edges are annotated with the amount of data to be transferred
  - Communication delay is considered during the evaluation of the design solution
- For cyclic task graph, we adopted the **Hierarchical Task Graph** representation, where nodes are classified as:
  - *simple*: tasks with no sub-tasks
  - *compound*: tasks with other HTGs associated (e.g., subroutines)
  - *loop*: tasks that represent a loop whose (partitioned) iteration body is a HTG itself

Example

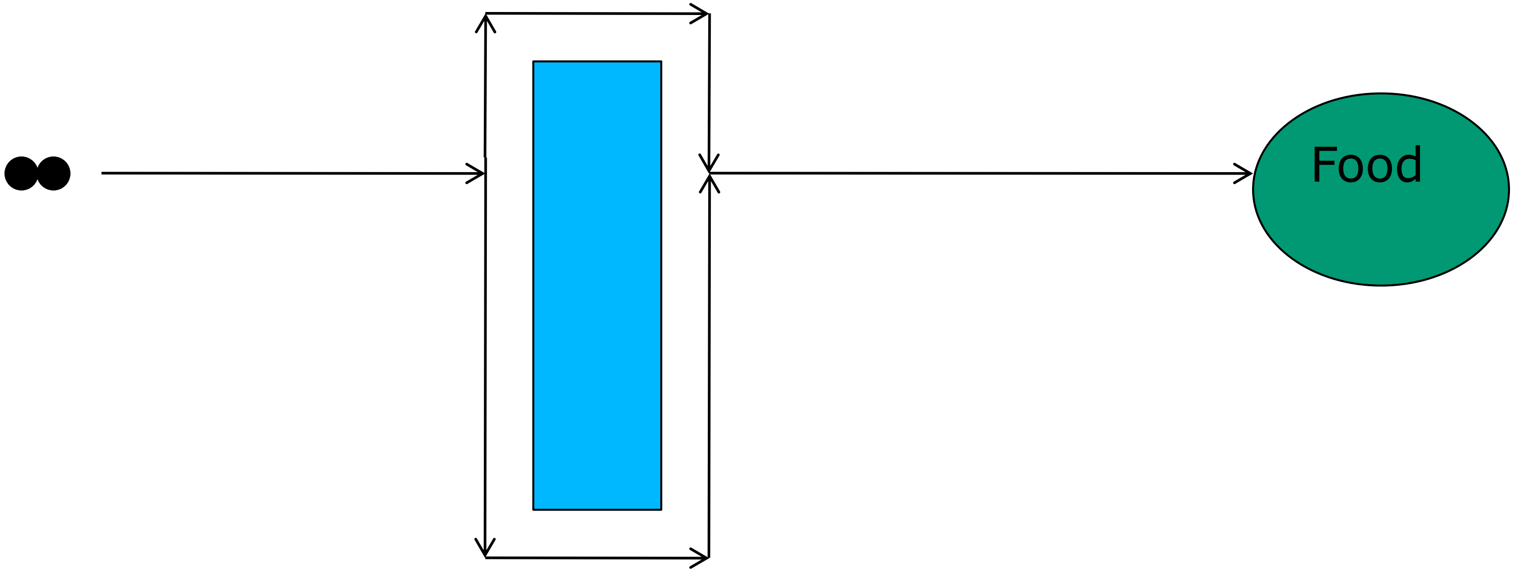
```
/* task T1*/
while(/*condition Loop0*/){
    #pragma omp parallel sections default(shared) num_threads(2)
    {
        #pragma omp section
        {
            while(/*condition Loop1*/){
              #pragma omp parallel sections default(shared) num_threads(2)
              {
                 #pragma omp section
                 {  /* task T2 */ }
                 #pragma omp section
                 {  /* task T3 */ }
              }
            }
        }
        #pragma omp section
        {
            while(/*condition Loop2*/){
              #pragma omp parallel sections default(shared) num_threads(2)
              {
                 #pragma omp section
                 {  /* task T4 */ }
                 #pragma omp section
                 {  /* task T5 */ }
              }
            }
        }
    }
}
/* task T6 */
```





Ant Colony Optimization (1)

- Introduced by Dorigo *et al.* as the Ant System (AS)
- Inspired by the observations of the behavior of ants when searching for food
- Ants start from their nest looking food going in random directions depositing a trail of pheromones that motivates other ants to follow the same path
- Cooperative behavior



Ant Colony Optimization (2)

- Initially formulated for the Traveling Salesman Problem
  - Find the best hamiltonian tour for all the cities (the nodes of a connected, undirected graph)

1. Associate each arc with a pheromone trail
2. Put m ants on an initial city
3. Each ant constructs its tour
4. The quality of the result is evaluated.
5. The pheromone trails are updated
6. If !goal, goto step 2.

- Probability decision:

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

- Pheromone update (std):

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

$$\Delta\tau_{ij}^{(l)} = Q/L$$

Problem Definition

- **Job**: generic activity (task or communication) to be completed in order to execute the specification.
- **Implementation point**: the mode for the execution of a job. It represents a combination of *latency* and *requirements of resources* on the related *target component*.
- **Mapping**: assign each job to an admissible implementation point, respecting the constraints imposed by the resources of the components.
- **Scheduling**: determine the order of execution of all the jobs of the specification in terms of priorities.
- **Objective**: minimize the overall execution time of the application on the target architecture.

Why ACO

- Ant Colony Optimization (ACO) heuristic is a constructive approach that limits as much as possible the generation of unfeasible solutions
  - Stochastic principles guarantee the exploration
  - Heuristic principles and feed-backs guarantee the exploitation of good parts of the solutions

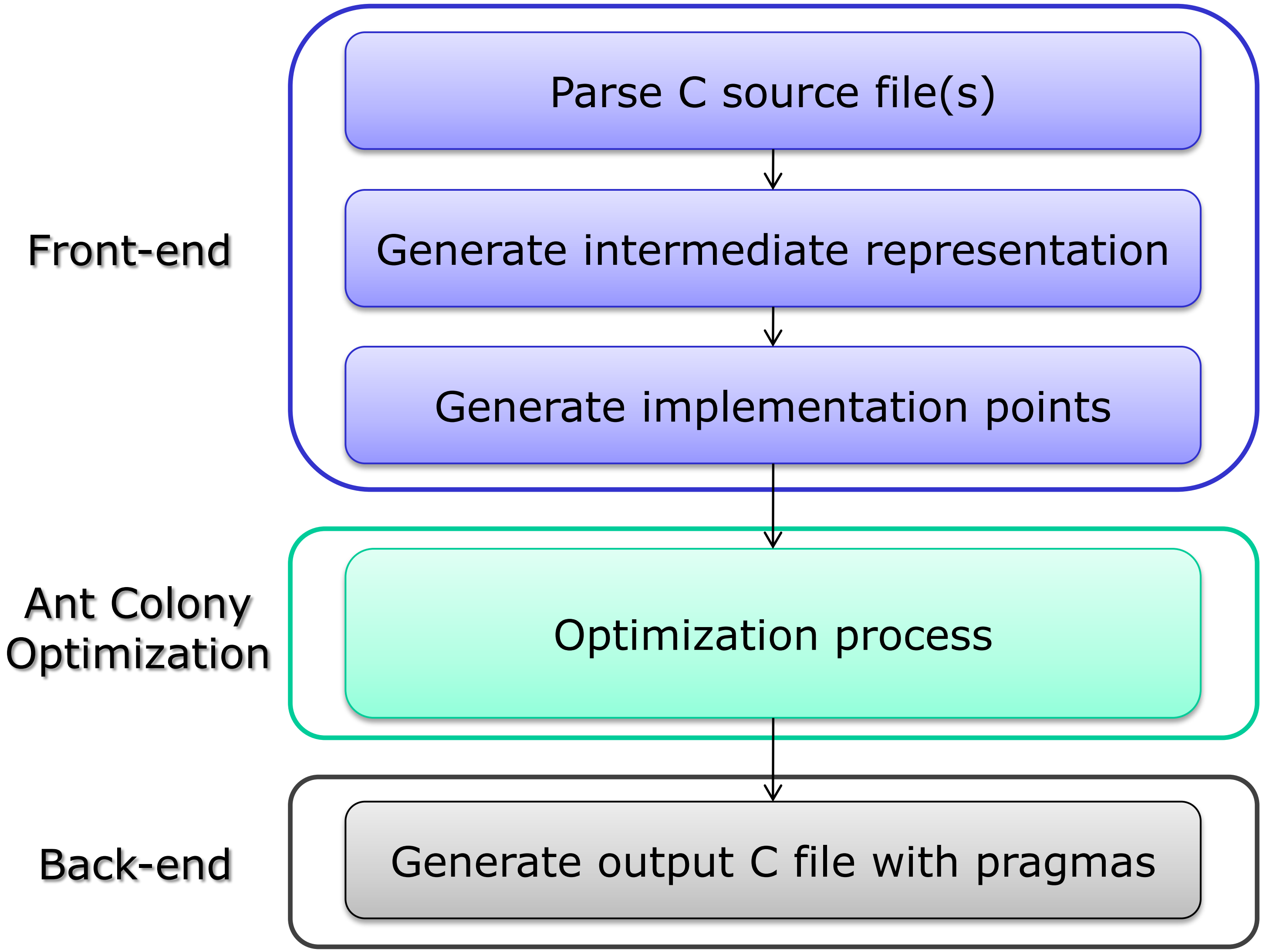
- Analysis and evaluation of different combination of mapping and scheduling

- Constructive approach, based on depth-first analysis, mimics the execution of the program and helps the handling of the design constraints, in particular with hierarchy.

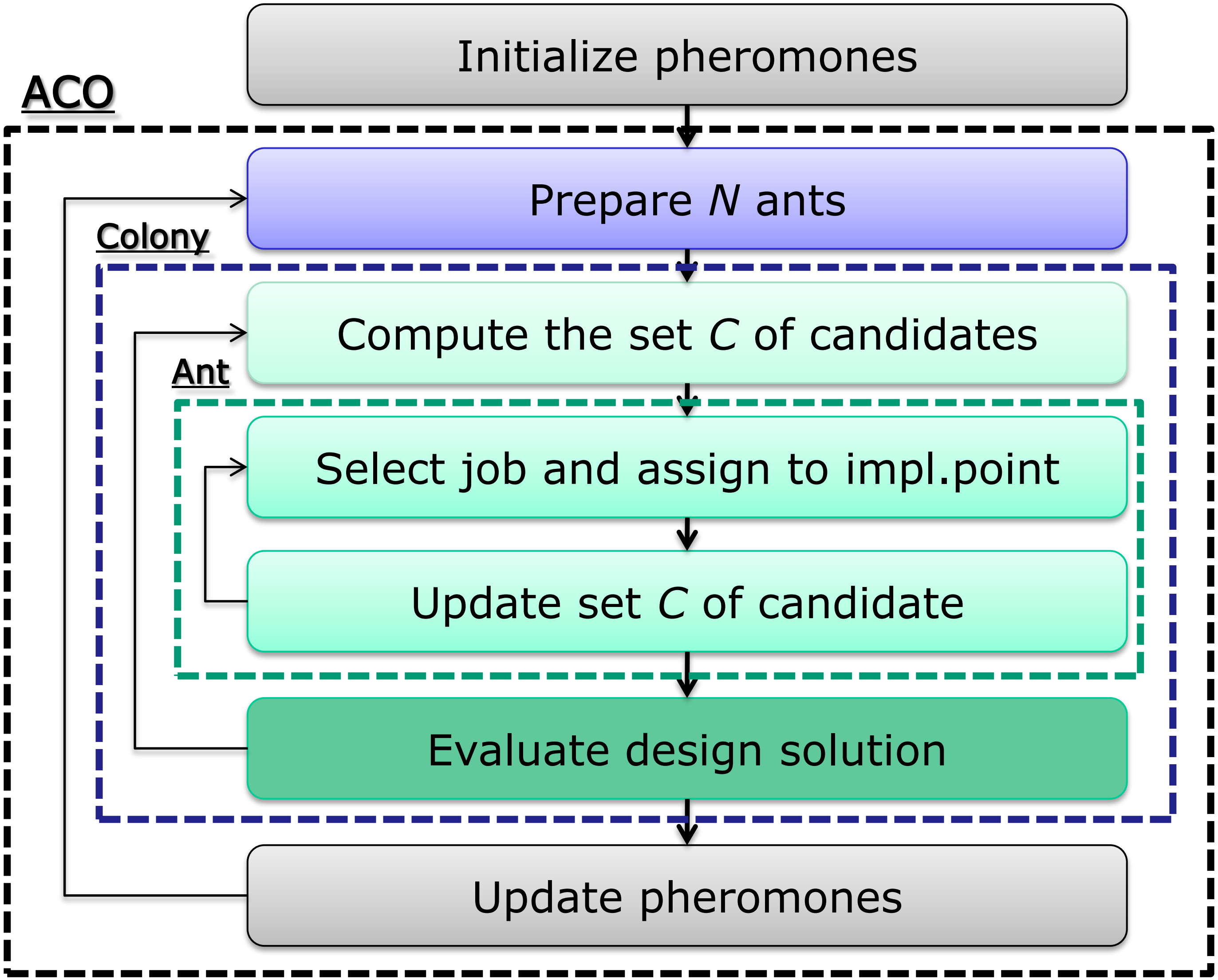
Why ACO

- Exploiting *resource partitioning* to avoid stalling the loops
- Most of the existing approaches rely on Direct Acyclic Graphs (DAGs)
  - Realistic C applications and (loop) partitioning are naturally described with cycles
- Function calls and loops introduce a hierarchy by definition
  - We maintain and exploit this hierarchy to better deal with the design constraints (top-level decisions influence low-level decisions)
  - A depth-first analysis on HTG is very similar to the actual execution of the application

Design Space Exploration



Design Space Exploration



Solution Evaluation

- The decisions performed by the ant give a **trace**
  - Sequence of jobs, where each of them is assigned to an implementation point
  - The position into the trace represents the priority for the scheduling (if they are selected early, they have higher priority…)
- List-based scheduler based on the mapping is given by the implementation point and the priority values
  - Different decisions performed by the ant correspond in exploring different design solutions (combination of mapping and scheduling)
- Return overall execution time of the application

Solution Evaluation for HTGs

- Evaluation for HTGs behaves as described before plus
  - At the same level of the hierarchy, tasks with higher priority are scheduled before tasks with lower priority
- If two parallel tasks (at the same level) have sub-graphs associated:
  - If the task **A** has higher priority than the task **B**, **A** is scheduled before **B**
  - Since a depth-first analysis is performed, the whole sub-graph associated to **A** is scheduled before the one associated to **B**
  - If the two sub-graphs do not involve the same processing elements, resource partitioning is exploited
- Average loop iterations improves the task-graph estimation

Experimental Results

- Comparison on DAGs with other common heuristics:
  - ACO requires very few evaluations to reach the optimum value
  - The number of unfeasible solutions is far reduced
  - The 2-stage decision process scales better with the size of the problem

#Tasks/ #Edges	#Degree	ILP		ACO 2-stage		ACO 1-stage		SA		TS		GA	
		Opt.	time (s)	time (s)	#eval.	time (s)	#eval.	time (s)	#eval.	time (s)	#eval.	time (s)	#eval.
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
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
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
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
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%)	
10/12	4	5,644	8.32	2.82	2,700	4.12	4100	(6,458 ± 2.48%)		6.87	25,093	3.33	10,798
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%)	
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%)	
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%)	
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%)	
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%))	
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)

Experimental Results: a case study

- The methodology has been successfully applied to the JPEG encoder on a multiprocessor prototype on FPGA
  - The heuristic is able to identify good design solutions for each different architecture configuration

2 PowerPC
1 Microblaze
8,400 free slices
1 shared bus

#Tasks/ #Edges	ACO Avg.	ACO %RSD	Platform Avg.	Platform %RSD	Diff.
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%

2 PowerPC
50 tasks
1 shared bus

MB/Area	ACO Avg.	ACO %RSD	Platform Avg.	Platform %RSD	Diff.
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%

- Prediction of the actual performance on the target platform is quite accurate

Experimental Results: HTGs

- Initial results on HTGs from MiBench suite:

Benchmark	ACO				SA		TS		Dyn.
	mix	cpu (s)	mapping	priority	mix	cpu (s)	mix	cpu (s)	sched.
sha	1.72 msec	4.20	+2.28 %	+12.14 %	+8.23 %	5.18	+6.71 %	7.11	+29.44 %
FFT	13.41 sec	8.12	+103.57 %	+108.38 %	+31.11 %	11.89	+27.84 %	17.20	+257.21 %
JPEG	0.46 sec	10.67	+0.12 %	+5.15 %	+1.13 %	14.63	+4.57 %	13.07	+27.64 %
susan	9.31 sec	6.08	+0.15 %	+21.96 %	+4.41 %	7.16	+7.58 %	9.18	+21.30 %
adpcm cod.	1.42 msec	0.20	+0.15 %	+7.08 %	+9.10 %	0.22	+4.33 %	0.25	+7.08 %
adpcm dec.	1.76 msec	0.19	+0.05 %	+4.65 %	+9.24 %	0.23	+8.96 %	0.21	+5.56 %
bitcount	0.15 sec	0.10	+1.12 %	+1,978.77 %	+11.02 %	0.10	+14.12 %	0.11	+2,024.77 %
	1.14 sec	0.34	+0.07 %	+178.07 %	+35.30 %	0.62	+29.89 %	0.58	+178.77 %
rijndael	0.81 sec	2.58	+2.12 %	+6.30 %	+3.12 %	3.36	+1.01 %	4.32	+3.40 %
	8.36 sec	2.72	+2.02 %	+6.20 %	+0.09 %	2.91	+0.74 %	3.10	+3.39 %
Avg. difference			+11.17 %	+232.87 %	+11.28 %	+27.53 %	+10.58 %	+45.21 %	+255.86 %

- It performs far better than SA, TS and Dynamic Scheduling*
- We are working on extending the ILP formulation to cyclic task graphs…

*Scheduling uses a FIFO policy - Mapping adopts a first available policy

Conclusions

- Ant Colony Optimization is very attractive for mapping and scheduling of C applications on heterogeneous MPSoCs
  - Constructive approach that limits unfeasible solutions
  - Handling of design constraints is very simple and efficient
- Results show that it is able to outperform most of the existing search methods
  - Very fast to reach the optimum value
  - Allocate and schedule efficiently also data transfers
  - Able to generate high-quality solutions in real-world applications
- Extensions to different communication models or architecture is straightforward.

Future Work

- Combining information from dynamic profiling has been demonstrated to improve the estimation of the task graph performance*

  * Fabrizio Ferrandi, Marco Lattuada, Christian Pilato, Antonino Tumeo, "*Performance Estimation for Task Graphs Combining Sequential Path Profiling and Control Dependence Regions*", In Proceedings of Seventh ACM-IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE'2009) [To Appear].

- In particular:
  - Dynamic path profiling gives information about the frequency of execution for all the paths into the specification
  - Analysis of the intermediate specification gives information of the contributions for all the tasks to each path
  - Estimation metrics for heterogeneous components based on machine learning techniques

Future Work

- Integrate the mapping phase with the task transformation phase
  - Mapping can support the clustering/partitioning of tasks
- Accurate estimation of communication latency and support to partial dynamic reconfiguration
- Concurrent optimization of application and architecture (system-level design)
  - ACO can suggest how to tailor the architecture to reduce the number of resources, without affecting the performance of the application

THANK YOU!