POLITECNICO DI MILANO

3rd Workshop on Mapping of Applications to MPSoCs

A Design Exploration Framework for Mapping and Scheduling onto Heterogeneous MPSoCs

Christian Pilato, Fabrizio Ferrandi, Donatella Sciuto
Dipartimento di Elettronica ed Informazione
Politecnico di Milano
{pilato,ferrandi,sciuto}@elet.polimi.it

Outline

- Introduction
  - hArtes project
  - Preliminaries and Problem Definition
- Proposed Methodology
  - Generation of Implemention Points
  - Application mapping and scheduling
- Experimental Results
- Conclusions and Future Work

hArtes Project

- Innovative FP6 European Project (2006-2010 – completed few weeks ago) to propose a 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 with respect to the target architecture
  - a system synthesis tool producing near-optimal implementations that best exploits the capability of each type of processing element

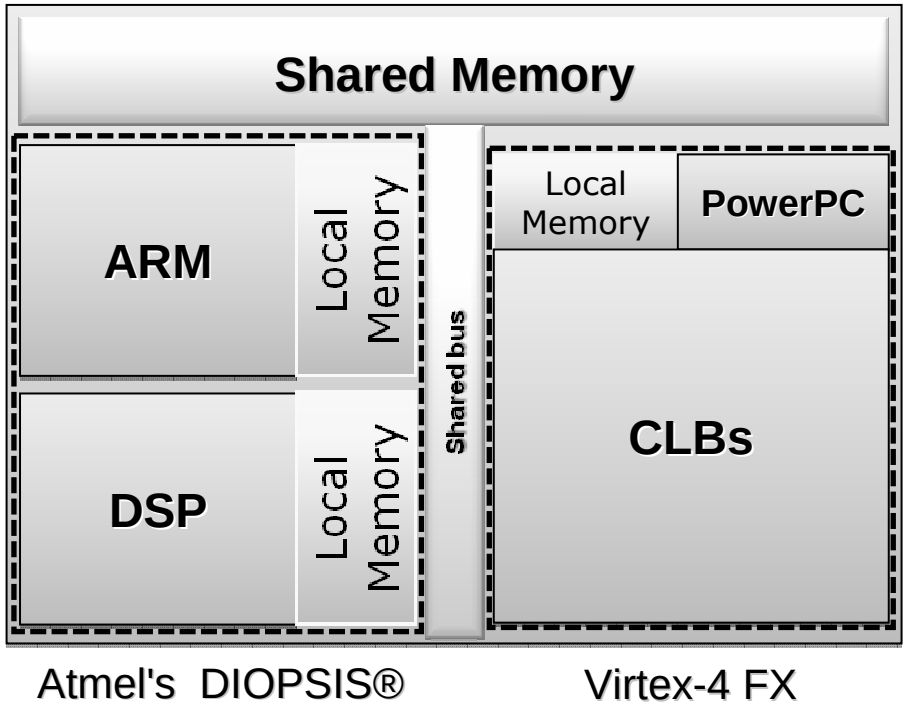
- We have in charge the automatic parallelization of the initial specification, performance estimation and an initial guess of mapping
  - C-to-C transformations and pragma insertion
  - The application has to be optimized with respect to a hardware architecture that is given

Problem Definition

- Porting a sequential application on a Multi-Processor System on Chip requires to:
  - Partition the application (*partitioning)*
    - Estimate the resource requirements of each task
    - Apply transformations to the task graph structure
  - 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 (e.g., limited area for HW devices)
  - Possibility to generate unfeasible solutions
- We need to model the **target architecture**, its **model of execution** and the **application**

Model of 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

Model of Execution

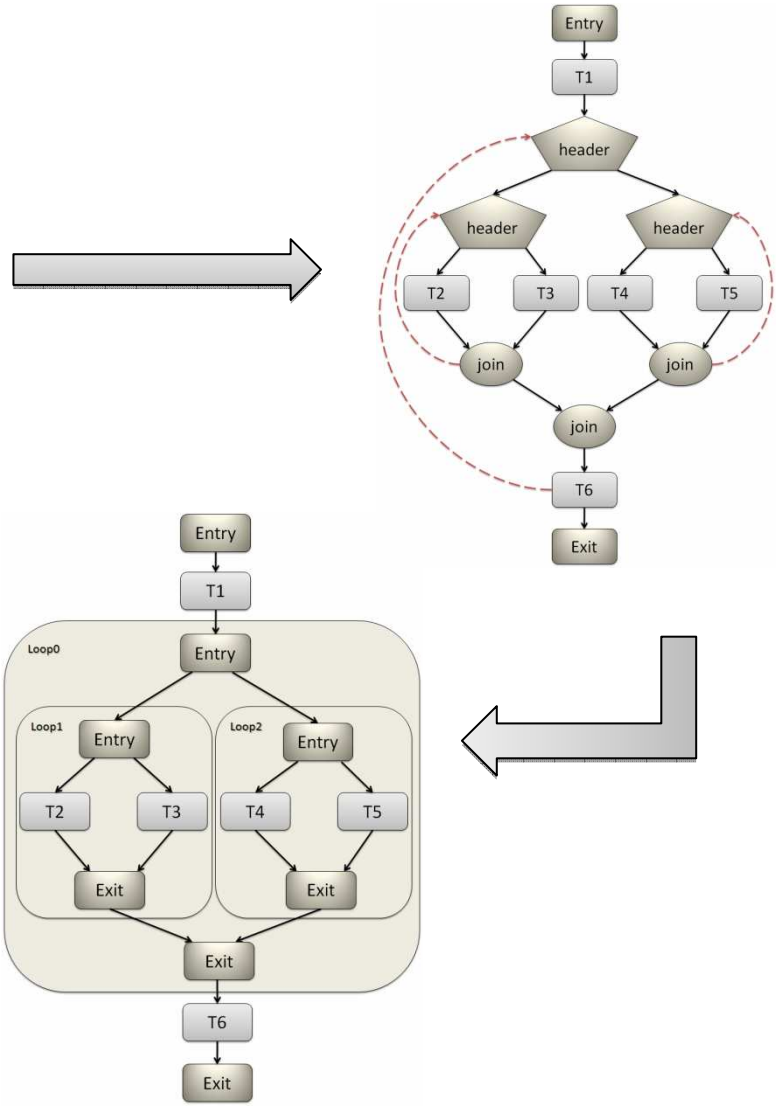
- The project relies on the MOLEN paradigm of execution
  - One master component (ARM) that manages the execution and starts other elements (*fork/join model*)
  - each task is represented through a function
- We adopted **OpenMP** as a standard for representing the partitioning inside the application
  - very simple and implicit notation to represent the fork/join model
  - validation can be also performed on the host machine (`-fopenmp`)
  - threads produced by `omp loop` are translated during the analysis into traditional tasks to be statically assigned to PEs
- Performance estimation and mapping work on standard C functions
  - estimation of the execution time of a C function
  - mapping of a sequence of function calls, where the structure of the task graph represents the parallelism

Model of Application

- Framework built upon the GNU/GCC compiler
  - we are able to control the optimizations and, then, exploit the resulting internal representation
- We represent the application through a **Hierarchical Task Graph**
  - Directly extracted from the C code annotated with OpenMP pragmas, where the hierarchy is induced by loops and functions
  - Nodes represent group of instructions and 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
  - Edges are annotated with the amount of data to be transferred
- Transformations to reduce the overhead required to manage the tasks based on a path-based estimation of the task graph [MEMOCODE '09]

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 */
```



Methodology Overview

Input

- Any C application (single source file of multiple source files)
  - Interfacing with GNU/GCC compiler
  - Annotated with pragmas associated to functions or parts of the code
    - OpenMP pragmas to described the partitioning
    - Profiling annotations, mapping suggestions, …
- XML file containing the description of the architecture and the implementation points, if available
  - Components, interconnections, sw/hw implementations, …

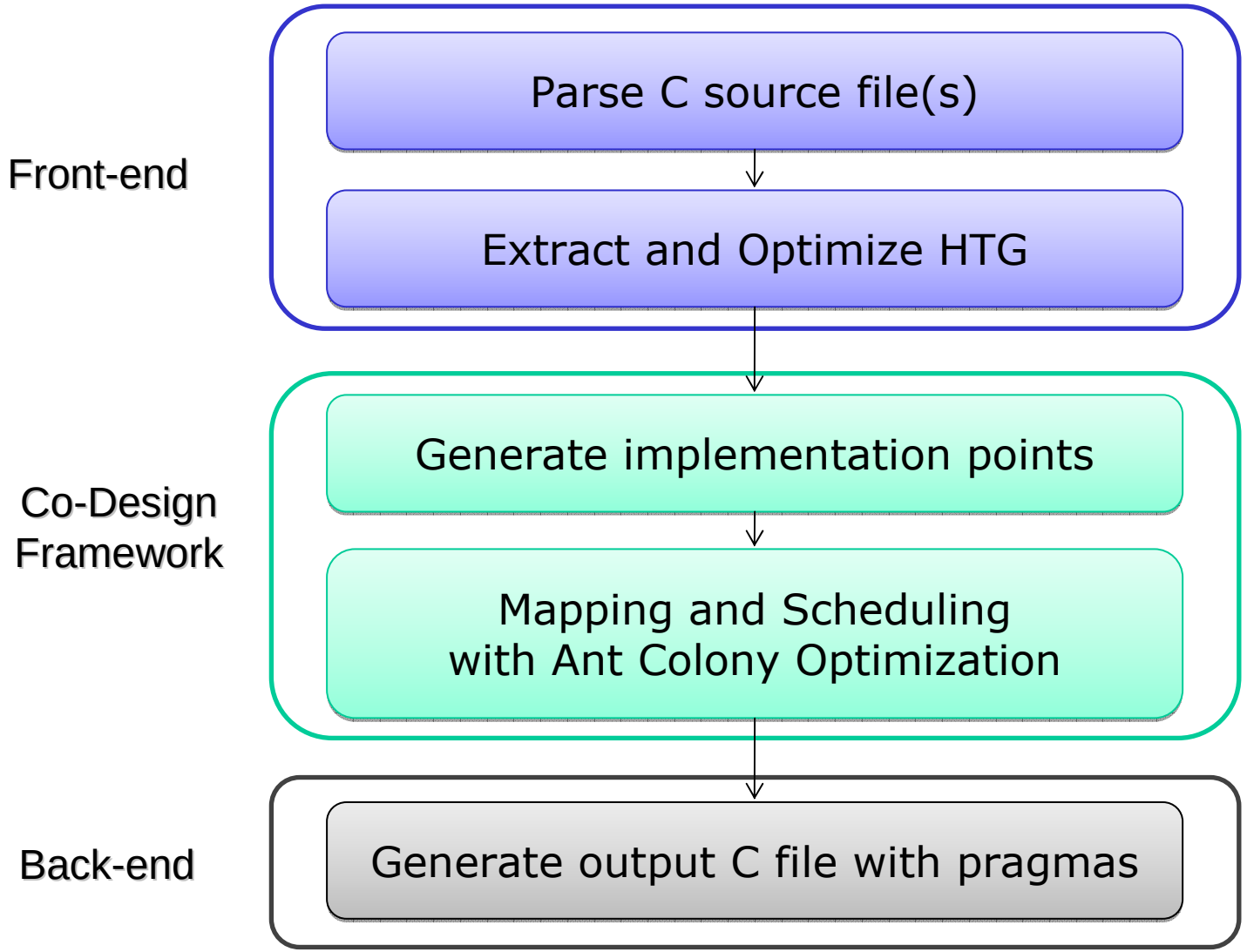
Output

- Tasks are represented as (new) functions
- C code, annotated with specific pragmas to represent the mapping decisions
- Priority table to represent the scheduling decisions

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.

Design Space Exploration



Generation of Implementation Points

- When not available into the XML file, we need to generate a "realistic" implementation point
  - Estimation of execution time and requirements of resources for each task on each component
- Different levels of accuracy for SW implementations
  - Code instrumentation and profiling on the target architecture
  - Estimations able to into account specific architectural characteristics of the processors (though a model based on linear regression and particular patterns – *sequences* – of operations) [CODES '10]
    - target independent representation (GIMPLE)
    - target dependent representation (RTL) – you need the compiler to expose it
- Design exploration framework for high-level synthesis to generate different Pareto-optimal HW implementations [JSA '08]
  - Multi-objective genetic algorithm (one objective for each resource)

Mapping and Scheduling with ACO

- Ant Colony Optimization (ACO) heuristic to analyze and evaluate different combinations of mapping and scheduling [ASPDAC '10]

- Constructive approach that limits as much as possible the generation of unfeasible solutions
  - Depth first-analysis that follows the hierarchy and mimics the execution of the program
  - Very simple to handle the different design constraints

- Combination of different principles to lead the exploration
  - Stochastic: at each step, a task is selected among the available ones and assigned to an implementation point through a roulette wheel extraction (exploration)
  - Heuristic: the probability is proportional to a combination of feedback information and a problem specific metric (exploitation)

Solution Evaluation

- The decisions performed by the ant give a **trace**
  - Sequence of jobs, where each of them is assigned to an implementation point (mapping)
  - The position into the trace represents the priority for the scheduling (if they are selected early, they have higher priority…)
  - Different traces correspond in exploring different design solutions (combination of mapping and scheduling)
- Evaluation performed though a list-based scheduler based on the mapping decisions and the priority values
  - Average loop iterations improves the task-graph estimation
- Return overall execution time of the application
  - Good solutions increase the feedback information of each decision
  - Bad solutions reduce the corresponding feedback information

Considerations for HTGs

- Additional considerations have to be introduced for HTGs
  - How to schedule tasks at different levels of hierarchy?
- Consider two parallel tasks **A** and **B** (at the same level), where each of them has a sub-graphs associated with:
  - The ant selects **A** before **B** (**A** has a higher priority than **B)**
  - During the evaluation, **A** is scheduled before **B**
  - Since a depth-first analysis is performed, the whole sub-graph (and the corresponding tasks) associated with **A** is scheduled before the one associated with **B**
  - If the two sub-graphs do not involve the same processing elements, resource partitioning is exploited and they can effectively run in parallel
    - we verified that the feedback information usually leads to such solutions, when possible

Handling of Design Constraints

- Avoid to allocate on non-renewable resource (e.g., FPGA area) tasks that cannot fit in the available area
  - The ant does not generate the related probability into the roulette wheel and the decision won't be taken for sure
- Limit the allocation of tasks that fork other tasks (e.g., containing function calls) to processing elements that cannot spawn threads (e.g., FPGA)
  - However, if allocated, all the sub-graph will be allocated to the same component (i.e., similar to *task inlining*)
- The implementation point of a task contains also information about the requirements of the sub-graphs, if any
  - Very simple to check if the current resources are able to satisfy the requirements if the subgraphs would have to be assigned to the same processing element

Handling of Design Constraints

- Hierarchy information (represented as a *stack*) helps the identification of candidate processing elements
  - Avoid to allocate tasks to processing elements occupied by higher level tasks
  - When there are not any *free* processing element, the task is executed by the current processing element
- When task migration is not supported, the decisions made for a function are replicated for all the instances (e.g., all the calls to the same function)
- Efficient and flexible representation for different communication models
  - A direct communication between local memories (e.g., though a DMA engine) is represented though a single job
  - A communication based on shared memory is represented though two jobs (from source task to local memory, from local memory to target task)

Experimental Results

- Results on HTGs from MiBench suite targeting a model of architecture very similar to hArtes's one

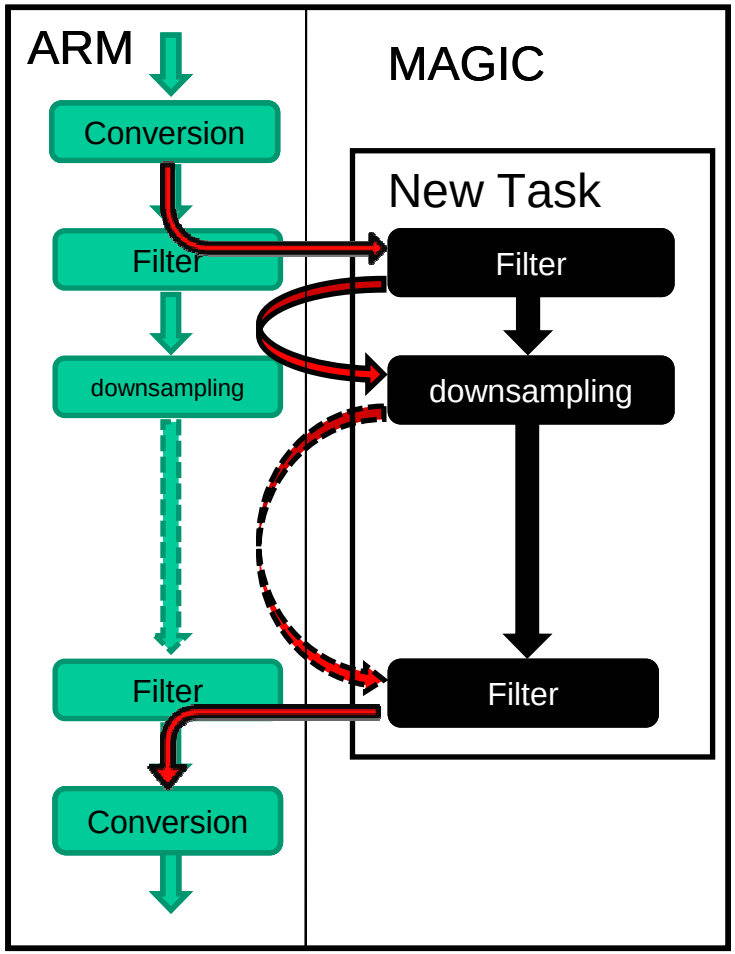
Benchmark	ACO				SA		TS		Dyn.
	mix	cpu (s)	mapping	priority	mix	cpu (s)	mix	cpu (s)	sched.
sha	1.72 msec	4.20	+2.28 %	+12.14 %	+8.23 %	5.18	+6.71 %	7.11	+29.44 %
FFT	13.41 sec	8.12	+103.57 %	+108.38 %	+31.11 %	11.89	+27.84 %	17.20	+257.21 %
JPEG	0.46 sec	10.67	+0.12 %	+5.15 %	+1.13 %	14.63	+4.57 %	13.07	+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*
  - The depth-first approach is more suitable to approach the problem
  - Much faster to coverge to a stable solution (not proven to be the optimum)
- We are working on extending the ILP formulation to cyclic task graphs…

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

Stationary Noise Filter

- Application provided by one of the partner of the project
- Execution time measured on the Atmel's DIOPSIS®
  - after executing each task, the control returns to the ARM
- Estimation techniques + the initial mapping
  - Speed-up: 4.3x
  - Overhead due to data transfers with ARM
- Applying transformations on the task graph
  - Speed-up: 5.8x
  - Unnecessary data transfers with ARM are removed



Conclusion and Future Works

- Flexible design exploration framework to map and schedule an OpenMP application on a given heterogeneous platform
  - Optimizes the HTG with iterative transformations
  - Estimates the implementation points for each task when not provided
  - Explores different combinations of mapping and scheduling

- Ongoing works
  - Co-exploration of partitioning and mapping into a unique loop
  - Co-exploration of application and target architecture
    - ReSP: open-source MPSoC simulation platform developed at Politecnico di Milano
    - FPGA prototyping platform based on different variants of Leon processors, Microblazes, PowerPCs, additional DSPs and HW cores

THANK YOU!

Christian Pilato
pilato@elet.polimi.it