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

Fabrizio Ferrandi, Christian Pilato, Donatella Sciuto, Antonino Tumeo

Politecnico di Milano – Dip. di Elettronica ed Informazione
{ferrandi,pilato,sciuto,tumeo}@elet.polimi.it
Outline

- Introduction
- Related Works
- Preliminaries and Motivation
- Proposed Methodology
  - Design Space Exploration with Ant Colony Optimization
  - Handling the Design Constraints
- Experimental Results
- Conclusions and Future Works
Mapping and scheduling of partitioned applications is crucial in particular for heterogeneous MPSoCs.

Most of existing approaches usually rely on DAGs (i.e., an acyclic representation).
- Difficulties to efficiently represent typical constructs in embedded applications (e.g., partitioned loops or function calls).

Different design constraints to be considered:
- Limited area for hardware devices, components that cannot spawn, preempt, migrate or switch threads, ...

Ant Colony Optimization (ACO) is a promising constructive method to produce very efficient solutions for the combined problem.
Related Works

- [Niemann and Marwedel 1997] **Exact solutions** for the combined problem with an ILP formulation on DAGs.
  - DAGs can be obtained through inlining and unrolling, but it greatly enlarges the design space and the complexity

- List-based scheduling is usually applied to obtain **heuristic solutions** based on priority information
  - [Beaty 1993], [Grajcar 1999] GA, TS and SA widely adopted to explore the best priority list
  - [Wiantong et al. 2002] The same search methods have been applied also to the mapping problem, but only on DAGs

- [Wang et al. 2005][Chang et al. 2008] ACO is becoming very attractive for such problems in recent years
Hierarchical Task Graph

- Given a DAG, delimiting the function regions or the loop body results in defining a sort of hierarchy into the graph
  - [Girkar and Polychronopoulos 1992] **Hierarchical Task Graph (HTG)**: intermediate representation for parallel programs

- An application can be represented by a HTG, where:
  - **Nodes** can be classified into:
    - *Simple*: tasks without sub-tasks (i.e., groups instructions to be sequentially performed)
    - *Compound*: tasks which consist of one or more HTGs, representing higher level structures, such as subroutines
    - *Loop*: tasks that represent a partitioned loop, whose iteration body is represented by a HTG itself
  - **Edges** represent the dependences among the tasks, annotated with the amount of data to be transferred
Generic architectural template composed of processing and communication elements. For example:

- **Shared Memory**
  - **ARM**
  - **DSP**
  - **MPSoC**
  - **CLBs**
  - **PowerPC**
  - **Virtex-4 FX**

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

- **Job**: generic activity (task or communication) to be completed in order to execute the specification

- **Implementation point**: combination of latency and requirements of resources for executing a job on a component

- **Mapping**: assign each job to an admissible implementation point, respecting the architectural constraints (e.g., the limited 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
Motivation

- Function calls and loops introduce a **hierarchy** by definition
  - HTGs maintain this hierarchy, helping to deal with 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

- **Ant Colony Optimization (ACO)** limits as much as possible the generation of unfeasible solutions
  - **Constructive approach**, based on depth-first analysis, helps the handling of the design constraints, specially with hierarchy.
  - Evaluation of **different combination** of mapping and scheduling
  - **Stochastic principles** guarantee the exploration
  - **Heuristic principles and feed-backs** guarantee the exploitation of good parts of the solutions
Methodology Overview

**Input**
- **Any C application** (single source file of multiple source files)
  - Interfacing with the GNU/GCC compiler (GIMPLE)
  - OpenMP pragmas to described the partitioning
  - Custom pragmas (e.g., profiling annotations, mapping suggestions)
- **XML file** describing the target architecture and the implementation points

**Output**
- **C code** annotated with **custom pragmas** to represent the mapping decisions
- **Priority table** to represent the scheduling decisions
Methodology Overview

Front-end

Parse C source file(s)

Generate HTG

Import implementation points

Design Space Exploration

Optimization process with ACO

Back-end

Generate output C file with pragmas and priority table
Design Space Exploration with ACO

1. Initialize pheromones
2. Prepare N ants
3. Compute the set $C$ of candidates
4. Select job and assign to impl. point
5. Update set $C$ of candidate
6. Evaluate design solution
7. Update pheromones
Stochastic Job Selection

- At each decision point \( (d) \), the probability to assign a candidate \( j \) to a proper implementation point \( i \):

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

  - The global heuristic represents the probability at the step \( d \) for the combination \( i,j \) to lead to a good solution

  - Roulette wheel and extraction of a combination of job and implementation point (mapping)

  - **Decision point** will correspond to the priority value
    
    - if selected early, they have higher priority...
Solution Evaluation

- List-based scheduler based on mapping decisions and priority values
  - Different ant decisions correspond in exploring different solutions

- At the same level of the hierarchy, tasks with higher priority are scheduled before tasks with lower priority
  - 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 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 (controlled by the heuristics!)

- Return overall execution time of the application
  - Feedback to compare different solutions
Handling of Design Constraints

- The implementation point of a task contains information also about sub-graphs
  - Useful when decisions at higher level imply decisions at lower level of the hierarchy (e.g., components that cannot spawn other threads)

- Avoid to allocate tasks on non-renewable resource (e.g., FPGA area) if they cannot fit in the available area
  - The ant does not generate the related probability and the decision will not be considered

- Constraint violations or unfeasible solutions can be easily identified
  - The corresponding decisions are penalized to avoid to be taken again in the future
Handling of Design Constraints

- Hierarchy information (as a stack) helps in identifying the candidate processing elements
  - If preemption and task switching are not supported, it avoids to allocate tasks to processing elements occupied by higher level tasks

- Limit as much as possible 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)

- When task migration is not supported, the decisions made for a function are replicated for all the instances (i.e., all the calls to that function)
Experimental Setup

- **Target architecture** composed of an ARM processor, a Digital Signal processor and an FPGA that also embeds a Power Pc processor
  - It allows to explore both hardware and software solutions
  - ARM processor is considered as the master that starts (and concludes) the execution of the applications
  - Only this processor can be interrupted, but just to manage the stitch code for the execution of the threads onto the other components, as well as the synchronizations
  - Partial dynamic configuration is not supported: tasks can be allocated to the FPGA as long as they fit into the available area

- Different embedded applications from **MiBench** suite manually partitioned with OpenMP pragmas and profiled
Experimental Results

- **Ant Colony Optimization**: our methodology

- **Search methods** (Simulated Annealing and Tabu Search)
  
  - permutation of the priorities and random changes of the mapping decisions

- **Dynamic scheduling**:
  
  - scheduling with a FIFO policy and mapping on first available

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>mix</th>
<th>cpu (s)</th>
<th>mapping</th>
<th>priority</th>
<th>mix</th>
<th>cpu (s)</th>
<th>mix</th>
<th>cpu (s)</th>
<th>Dyn. sched.</th>
</tr>
</thead>
<tbody>
<tr>
<td>sha</td>
<td>1.72 msec</td>
<td>4.20</td>
<td>+2.28 %</td>
<td>+12.14 %</td>
<td>+8.23 %</td>
<td>5.18</td>
<td>+6.71 %</td>
<td>7.11</td>
<td>+29.44 %</td>
</tr>
<tr>
<td>FFT</td>
<td>13.41 sec</td>
<td>8.12</td>
<td>+103.57 %</td>
<td>+108.38 %</td>
<td>+31.11 %</td>
<td>11.89</td>
<td>+27.84 %</td>
<td>17.20</td>
<td>+257.21 %</td>
</tr>
<tr>
<td>JPEG</td>
<td>0.46 sec</td>
<td>10.67</td>
<td>+0.12 %</td>
<td>+5.15 %</td>
<td>+1.13 %</td>
<td>14.63</td>
<td>+4.57 %</td>
<td>13.07</td>
<td>+27.64 %</td>
</tr>
<tr>
<td>susan</td>
<td>9.31 sec</td>
<td>6.08</td>
<td>+0.15 %</td>
<td>+21.96 %</td>
<td>+4.41 %</td>
<td>7.16</td>
<td>+7.58 %</td>
<td>9.18</td>
<td>+21.30 %</td>
</tr>
<tr>
<td>adpcm cod.</td>
<td>1.42 msec</td>
<td>0.20</td>
<td>+0.15 %</td>
<td>+7.08 %</td>
<td>+9.10 %</td>
<td>0.22</td>
<td>+4.33 %</td>
<td>0.25</td>
<td>+7.98 %</td>
</tr>
<tr>
<td>adpcm dec.</td>
<td>1.76 msec</td>
<td>0.19</td>
<td>+0.05 %</td>
<td>+4.65 %</td>
<td>+9.24 %</td>
<td>0.23</td>
<td>+8.96 %</td>
<td>0.21</td>
<td>+5.56 %</td>
</tr>
<tr>
<td>bitcount</td>
<td>0.15 sec</td>
<td>0.10</td>
<td>+1.12 %</td>
<td>+1,978.77%</td>
<td>+11.02 %</td>
<td>0.10</td>
<td>+14.12 %</td>
<td>0.11</td>
<td>+2,024.77%</td>
</tr>
<tr>
<td>rijndael</td>
<td>1.14 sec</td>
<td>0.34</td>
<td>+0.07 %</td>
<td>+178.07%</td>
<td>+35.30 %</td>
<td>0.62</td>
<td>+29.89 %</td>
<td>0.58</td>
<td>+178.77 %</td>
</tr>
<tr>
<td></td>
<td>0.81 sec</td>
<td>2.58</td>
<td>+2.12 %</td>
<td>+6.30 %</td>
<td>+3.12 %</td>
<td>3.36</td>
<td>+1.01 %</td>
<td>4.32</td>
<td>+3.40 %</td>
</tr>
<tr>
<td></td>
<td>8.36 sec</td>
<td>2.72</td>
<td>+2.02 %</td>
<td>+6.20 %</td>
<td>+0.09 %</td>
<td>2.91</td>
<td>+0.74 %</td>
<td>3.10</td>
<td>+3.39 %</td>
</tr>
</tbody>
</table>

| Avg. difference | +11.17 % | +232.87 % | +11.28 % | +27.53 % | +10.58 % | +45.21 % | +255.86 % |

Christian Pilato – Taipei (Taiwan) – January 21st, 2010
Conclusions

- Results show that ACO is able to outperform most of the existing methods
  - Very fast to reach good solutions with respect to other methods
  - Able to generate high-quality solutions in real-world applications

- ACO is very attractive for mapping and scheduling of parallel C applications on heterogeneous MPSoCs
  - The **depth-first approach** is more suitable to approach the problem
  - Limiting the unfeasible solutions, it has **better elaboration time** (i.e., it does not get stuck to exit from unfeasible regions)
  - **Handling of design constraints** is very simple and efficient

- Extensions to consider different communication models is straightforward
Future Work

- Estimation metrics for heterogeneous components based on machine learning techniques

- Combining information from dynamic profiling improves the estimation of the task graph performance*

- A fast estimation of the tasks’ annotations and task graph performance opens new possibilities for automatic parallelizing compilers
  - Task transformation methodologies that are aware of the final target architecture for both parallelization and mapping into a unique loop

Research partially funded by the European Community’s Sixth Framework Programme, hArtes project (http://www.hartes.org)

ANY QUESTION?

THANK YOU!

pilato@elet.polimi.it