A Synthesis Approach for Mapping Irregular Applications on Reconfigurable Architectures

Vito Giovanni Castellana, Antonino Tumeo, Fabrizio Ferrandi

Pacific Northwest NATIONAL LABORATORY
Proudly Operated by Battelle Since 1965

Motivation

Emerging applications such as bioinformatics and knowledge discovery are irregular: they use data structures based on pointers or linked lists (graphs, trees, grids), which generate **unpredictable memory accesses**

They are commonly
- Memory bandwidth bounded, with high synchronization intensity
- Highly parallel, at task level granularity

Novel approaches aim at mapping irregular applications on reconfigurable platforms, accelerating them through custom hardware
- Example: hybrid architectures (e.g. Convey HC1, HC2, MX100 platforms)

Existing High Level Synthesis (HLS) tools are able to automatically generate hardware implementations starting from C/C++ code, but …

Hardware designers intervention is still needed for the development of both custom accelerators and (complex) synchronization structures

Problem Definition

Objective

Design of an **automated flow** for the synthesis of custom hardware components, focused on accelerating irregular applications

Applications are modeled through a general template, able to map a large class of irregular algorithms, such as graph exploration

```
void application_template(){
    //code block

    for(id=init; id<NUM_it; id=id+1 ){
        kernel(id, data);
    }
    //code block
}
```

Challenges

Exploit coarse grained parallelism
- **Several kernels may run concurrently**

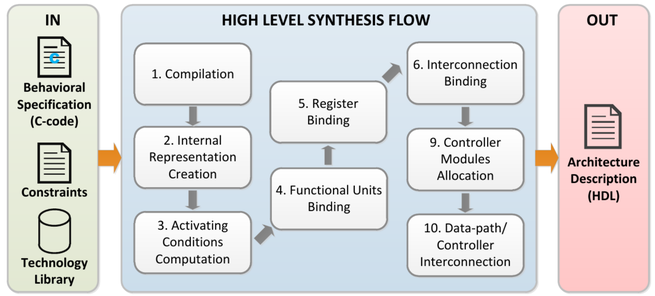
Kernels perform independently unpredictable memory accesses
- **Structural conflicts have to be avoided**

Kernels may communicate through the shared memory
- **Synchronization directives (e.g. atomic operations) should be implemented**

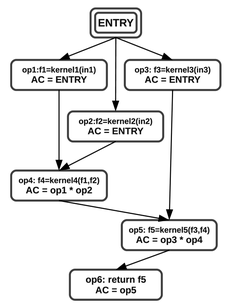
High Level Synthesis Flow

HLS framework for the synthesis of hardware components featuring dynamic scheduling
- Extends the Bambu HLS framework, available at **panda.dei.polimi.it**

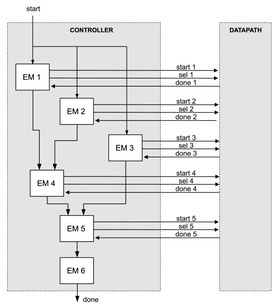
Input: a C-code description of the specification
Output: the corresponding HDL implementation, ready for synthesis



- Source code compilation (1) → graph IR (2)
- IR analysis: Activating Conditions (ACs) identifications(3)
  - ACs: logic expressions which state when an operation can be executed
  - Their satisfaction is checked at runtime through dedicated hardware
- Data-path synthesis: registers(4), functional units(5) and interconnections (6) binding
- Controller generation (7)
  - Controller designed as a set of simple sub controllers, each associated to an operation;
  - it allows to manage concurrent flows of execution ...
  - … including **multiple parallel kernels**



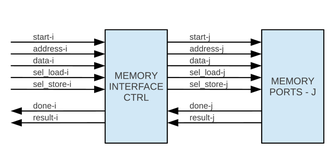
Example IR, annotated with activating conditions



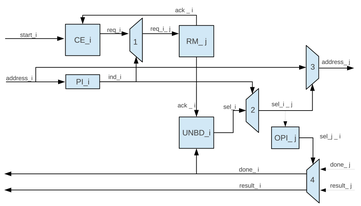
Schematic of the resulting accelerator architecture

Memory interface Controller (MIC)

It allows multiple hardware components to interface with shared memory banks. The MIC:
- Dynamically maps **unpredictable** memory accesses to memory ports, elaborating at runtime the corresponding addresses
- Manages **concurrency**: it collects memory requests, and if their target addresses collide, it forwards them one at a time
- Directly implements **atomic operations** (e.g. atomic increment, compare and swap) through dedicated hardware



The MIC dynamically forwards I memory access towards J memory banks



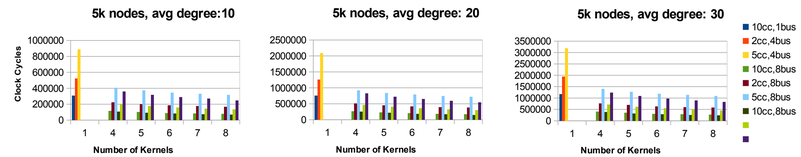
Basic MIC structure, for i-th inputs; it elaborates the input address, and routes the memory access accordingly

The MIC does not introduce any delay penalty, serving requests as soon as the memory resources are available

Experimental Validation

Performances (execution latency, area) evaluated through the synthesis of the Breath First Search, varying the number of allocated kernels and number of memory banks

- Increasing the number of both kernels and memory banks provide valuable speed ups



Simulation results: execution latencies varying the size of the input graph (randomly generated)

	1kernel/1out	4kernel	5kernel	6kernel	7kernel	8kernel
FF,4out	942	2466	3066	3415	4124	4563
LUT,4out	1141	4981	6200	7218	8378	8911
FF,8out	942	2611	3210	3559	4268	4706
LUT,8out	1141	6367	7912	9305	10935	11586

Area evaluation: number of Flip Flop (FF) registers and Look Up tables (LUTs) required to implement the generated designs

Vito Giovanni Castellana, Fabrizio Ferrandi
Politecnico di Milano, DEIB
vcastellana@elet.polimi.it, ferrandi@elet.polimi.it

Vito Giovanni Castellana, Antonino Tumeo
Pacific Northwest National Laboratory, HPC
vitogiovanni.castellana@pnnl.gov, antonino.tumeo@pnnl.gov