A Novel Design Framework for the Design of Reconfigurable Systems based on NoCs

Vincenzo Rana, Ivan Beretta, Donatella Sciuto

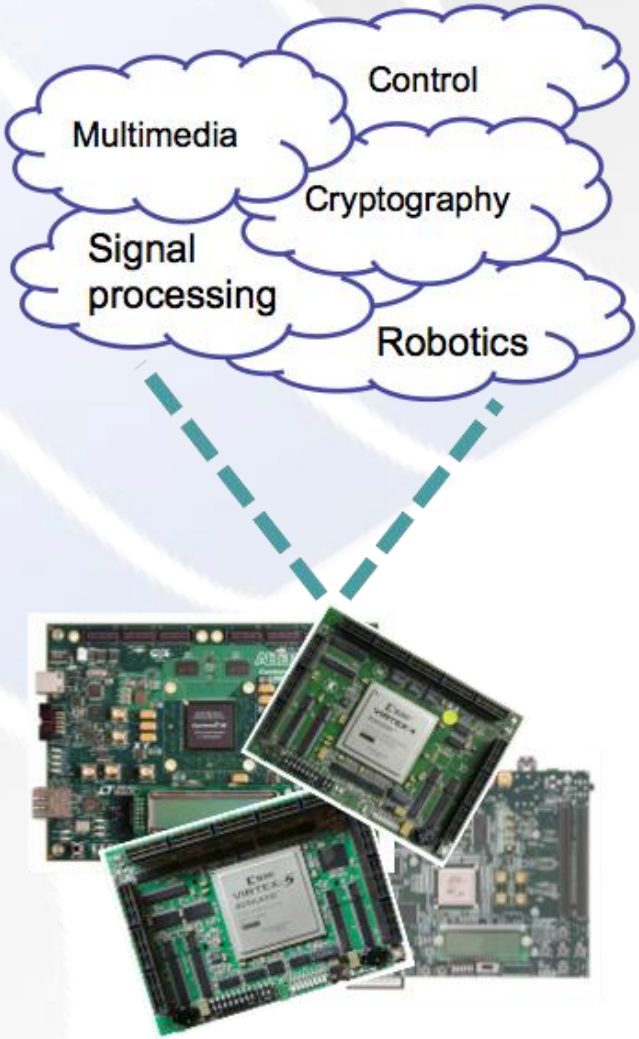
Donatella Sciuto
sciuto@elet.polimi.it

Outline

- Introduction
- Related Work
- Innovative Contributions
- Basic Concepts
- Network-on-Chip
- Mapping of cores on NoCs
  - at Design-Time
  - at Run-Time
- Experimental Results
- Concluding Remarks

Introduction

- Increasing popularity of multi-core applications
  - Multi-core processing and on-demand acceleration
- Need for flexibility on the hardware side
  - Field Programmable Gate Arrays (FPGAs)
  - Networks-on-Chip (NoCs)
  - Partial dynamic reconfiguration
- Need for reconfiguration-aware design methodologies
  - CAD tools to tackle dynamic reconfiguration
- Application mapping
  - Efficient assignment of each core to a specific FPGA region…
  - … Even at run-time



Related Work: NoC

- XPIPES [Bertozzi et al.]
  - static NoC
  - low area usage
    - from 86 to 267 slices for a single switch
  - very good timing performance
    - single switch latency (clock cycles): 1
    - single switch latency (ns): 5.9

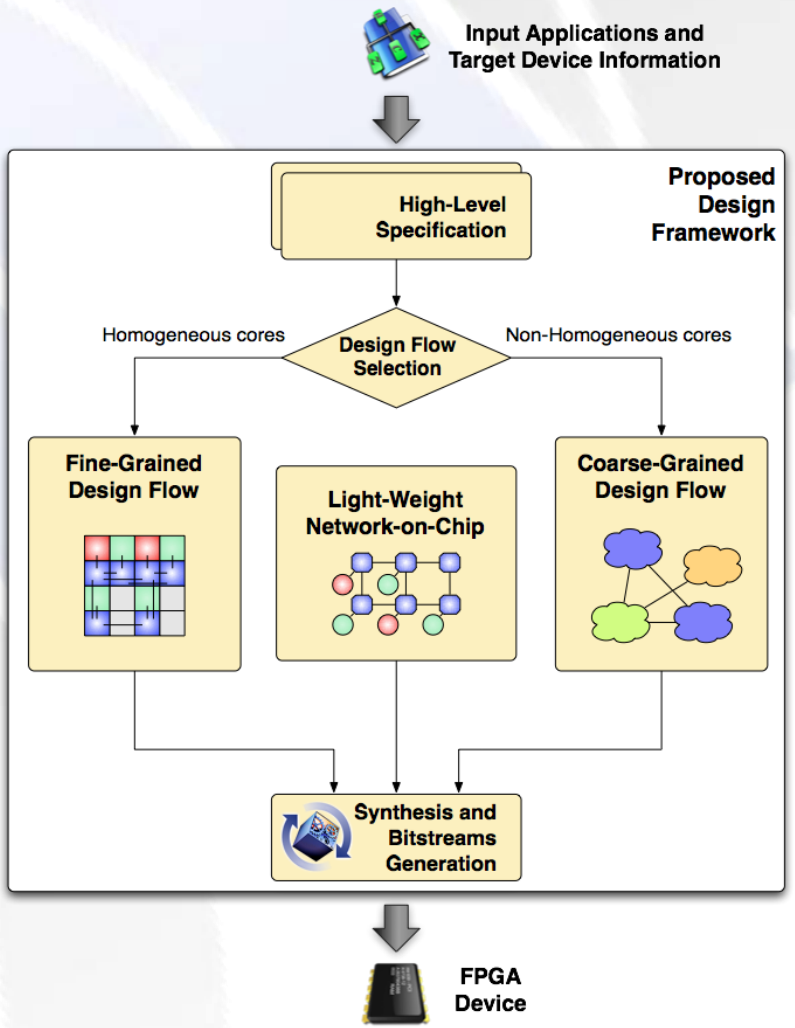
- CoNoChi [Pionteck et al.]
  - fully reconfigurable NoC
  - quite high area usage
    - from 363 to 493 slices for a single switch
  - bad timing performance
    - single switch latency (clock cycles): 5
    - single switch latency (ns): from 45 to 76

Related Work: Mapping Algorithms

- Mapping of computation cores on NoC-based systems
  - To optimize communication overhead [Murali and De Micheli], area, power consumption [Murali et al.], network size [Hansson et al.]
  - Do not explicitly handle dynamic reconfiguration
  - Can only be executed at design time
- Incremental mapping of new applications in an executing system [Chou et al.]
  - All the applications are concurrently mapped on the device
- Mapping of a single application on a reconfigurable device [Ghiasi et al.]
  - The application must satisfy a strict set of constraints
- Our approach allows dynamic addition of new applications and exploits dynamic reconfiguration

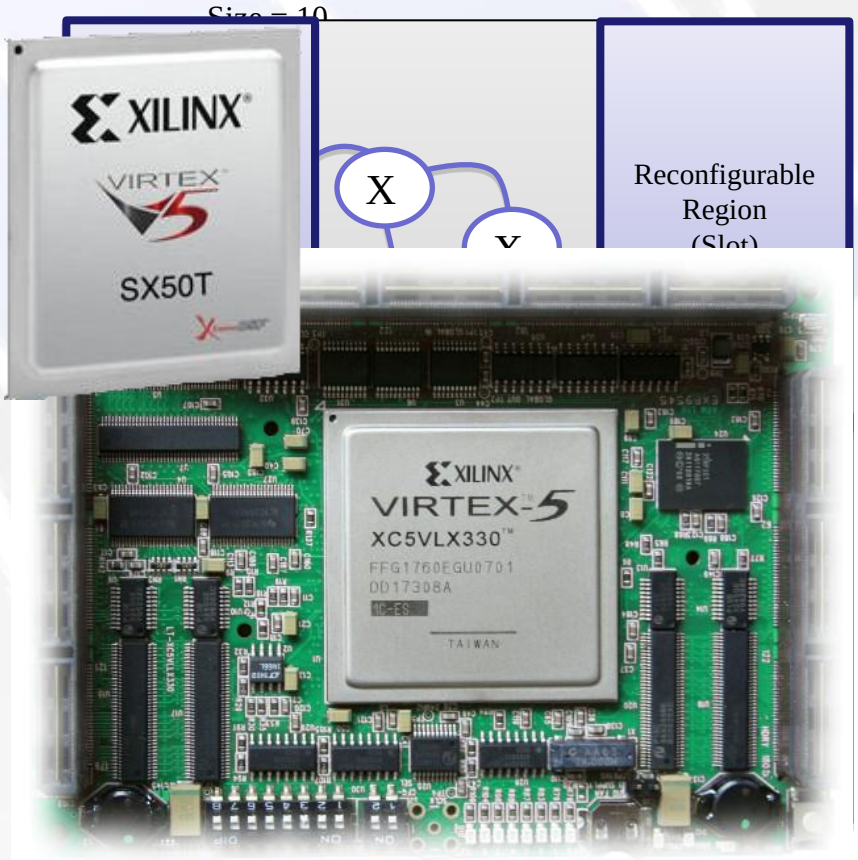
Innovative Contributions

- Light-Weight NoC
  - Reconfigurable NoC
  - Hybrid protocol able to fully support reconfigurations
  - Very-high performance
- Design Framework
  - Complete Design Framework for reconfigurable systems
    - From high-level specification to bitstreams
  - Two Design Flows for the minimization of
    - communication overhead
    - reconfiguration overhead



The Proposed Approach

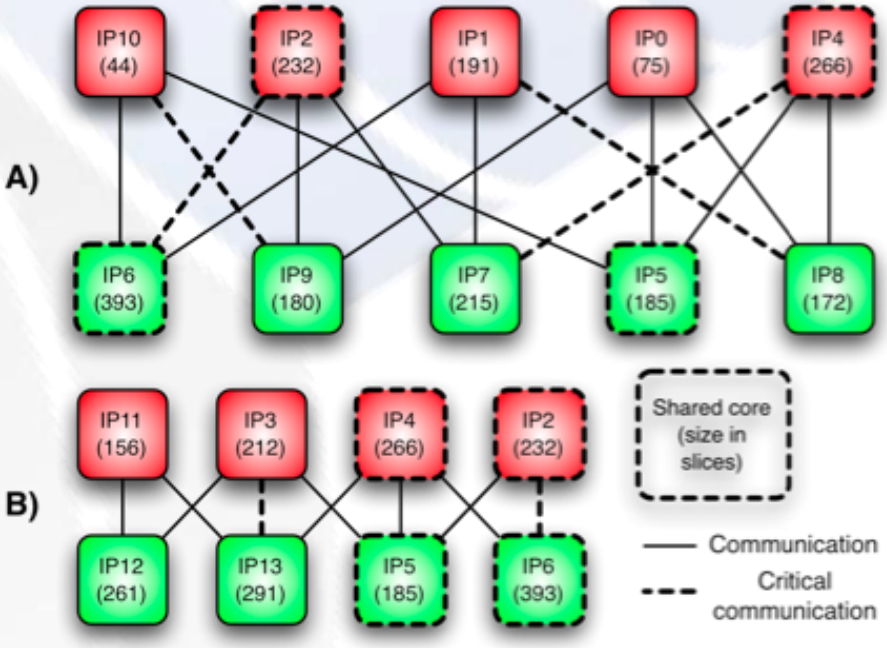
- Application
  - Multiple cores that cooperate to achieve a task
  - Communication graph
- Target device
  - New FPGA families (e.g. Xilinx Virtex-4 and 5)
  - Switching among multiple applications
- Hardware architecture
  - Network-on-Chip (NoC)
  - Fixed-size slots



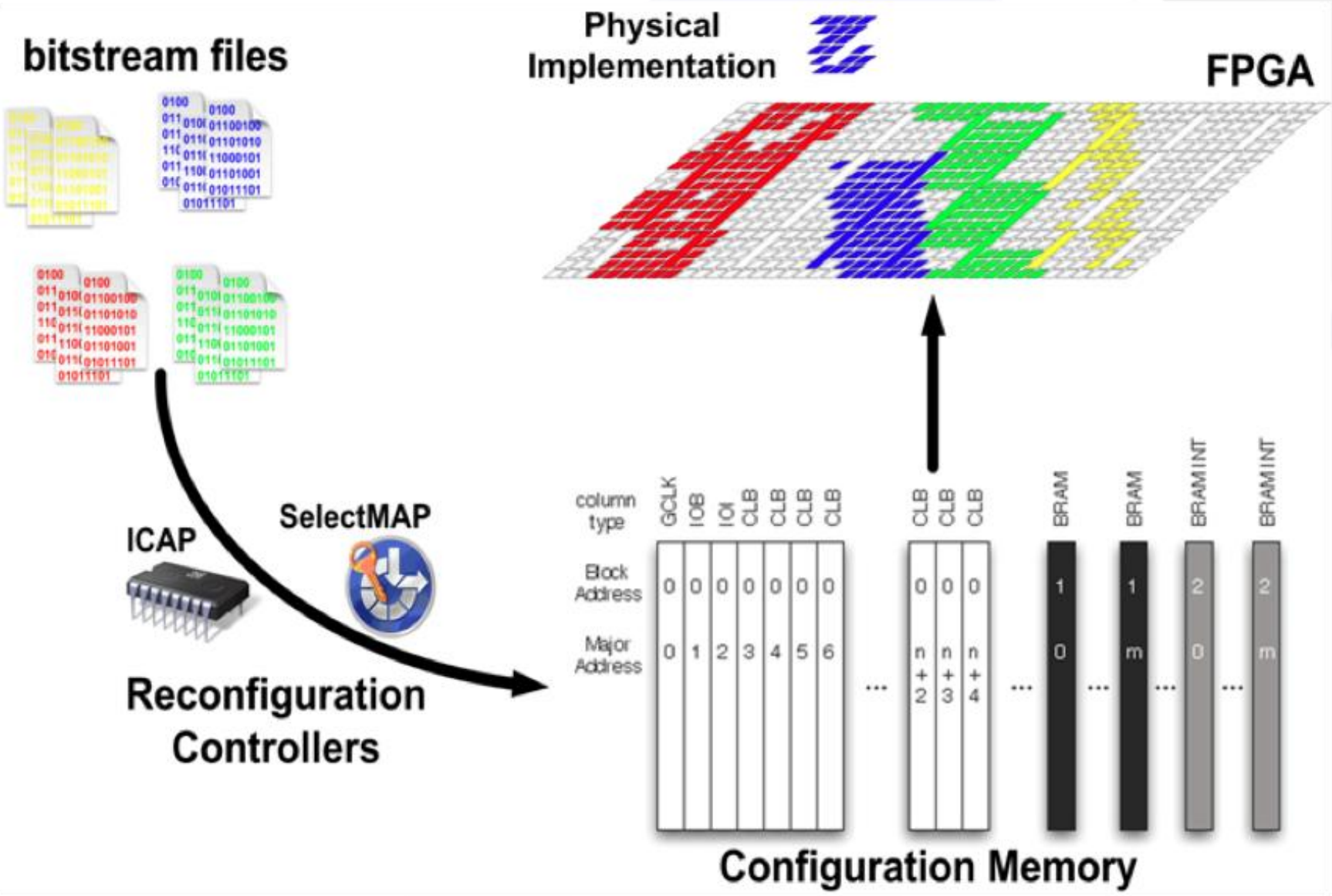
Applications and CGs

- Multiple applications can be executed on an FPGA
  - Either at the same time or in different time slots
- Each application needs several soft cores that need to be configured on the device
  - The communication constraints

and requirements of these applications can be represented through Communication Graphs (CGs)

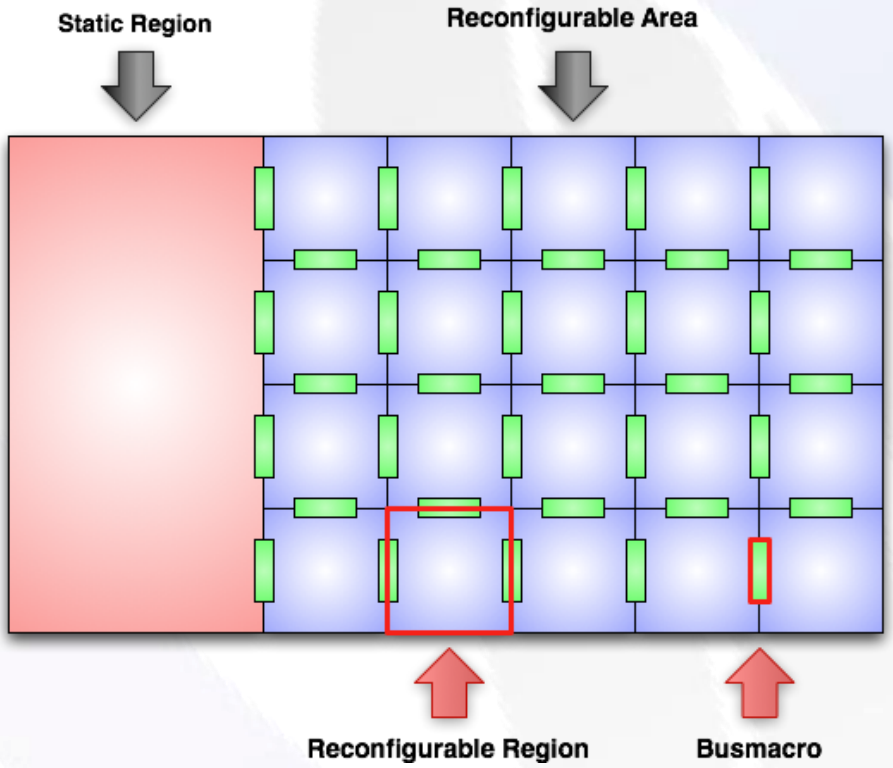


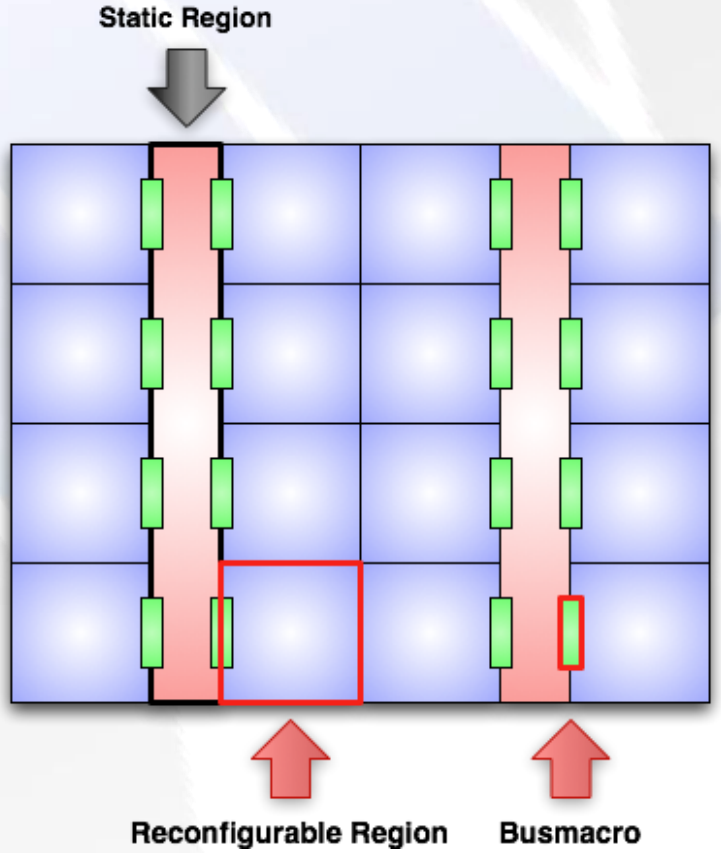
Dynamic Reconfiguration



Target Architecture

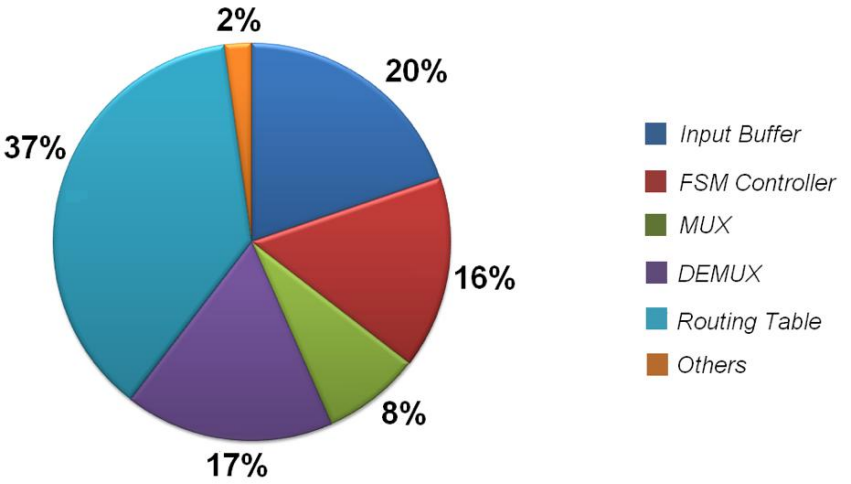
- FPGA-based reconfigurable embedded systems
  - Static Regions
  - Reconfigurable Regions
  - Busmacro

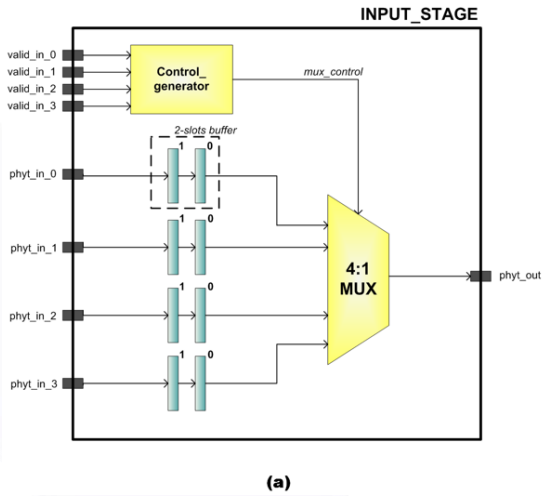


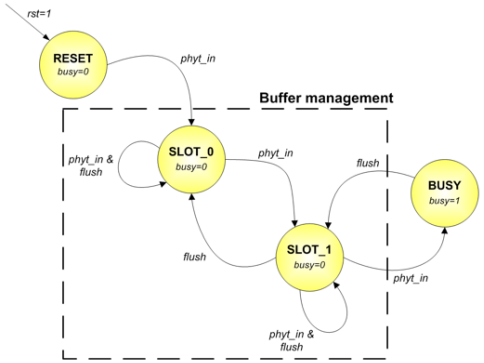


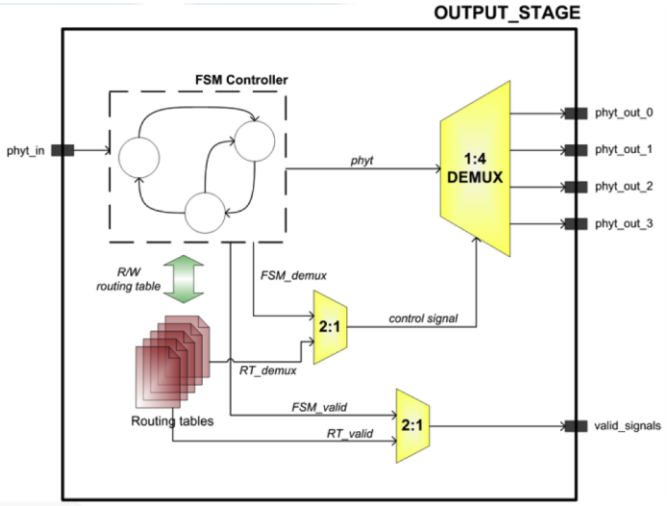
NoC implementation (1/2)

- Switches
  - Input Stage
    - buffers
  - Output Stage
    - routing tables



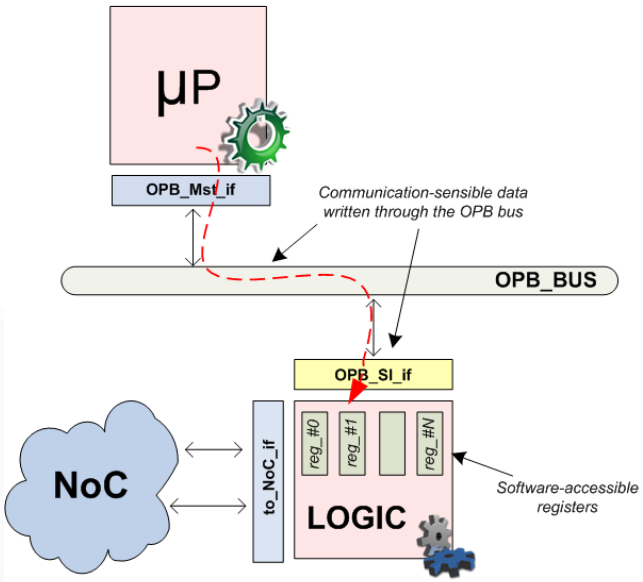


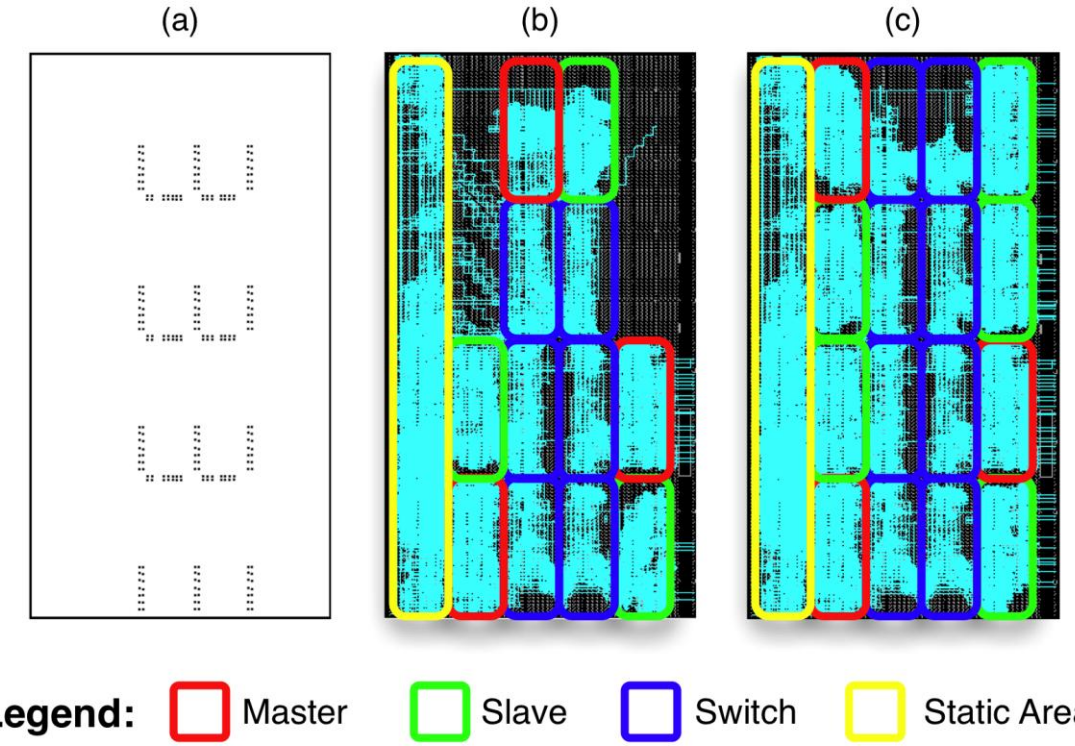




NoC implementation (2/2)

- Network Interfaces
  - Target and Initiator NIs
  - On-chip Peripheral Bus (OPB), Processor Local Bus (PLB)
- NoC Protocol (hybrid)





NoCs results comparison

- XPipes
  - Not reconfigurable
  - Area usage for a single switch: from 86 to 267 slices
  - Single switch latency: 5.9 ns
- CoNoChi
  - Fully reconfigurable
  - Area usage for a single switch: from 363 to 493 slices
  - Single switch latency: from 45 to 76 ns
- The proposed Light-Weight NoC
  - Fully reconfigurable
  - Area usage for a single switch: from 224 to 308 slices
  - Single switch latency: from 2.7 to 6.1 ns

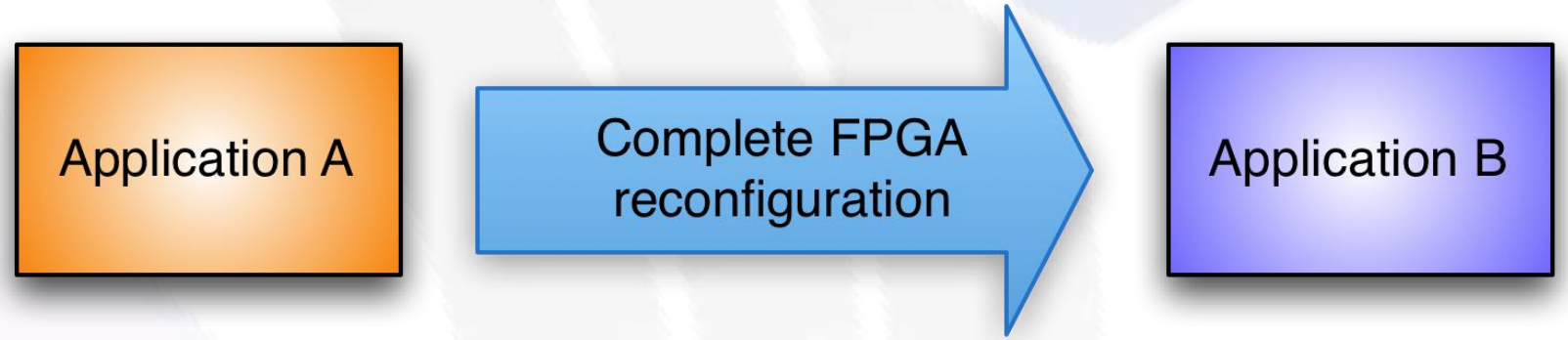
Mapping cores on the NoC

- In order to fully exploit the potential of the NoC architecture it is necessary to perform a mapping between the soft-cores and the network switches
  - minimizing the distance among the cores that communicate the most
  - avoiding congestion on the links between each couple of network switches
- The communication constraints to be satisfied could be different from application to application
  - each application could potentially require a different mapping of the cores on the NoC

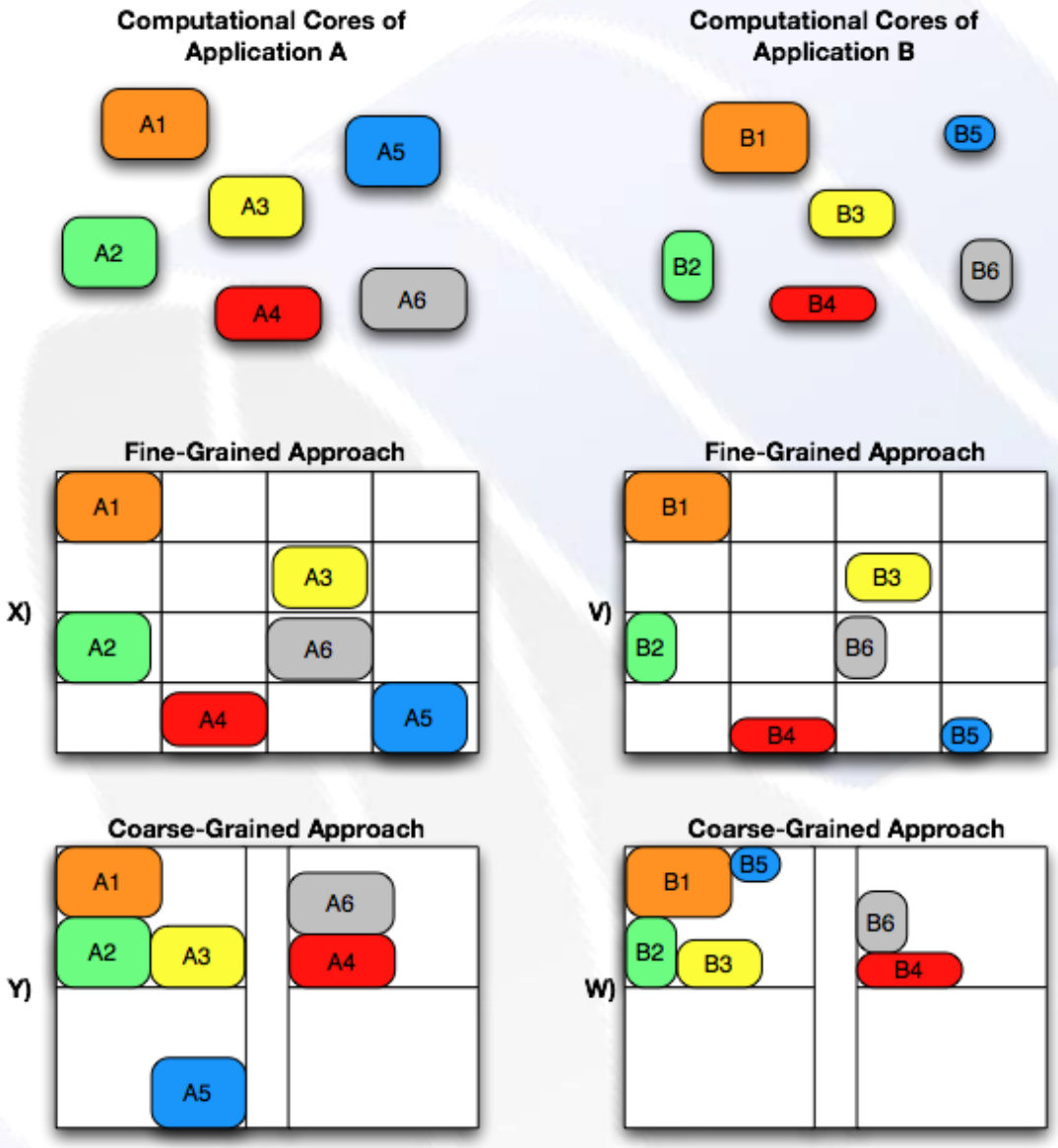
Let's do it simple

- Complete reconfiguration
  - an optimized synthesis for application A
  - an optimized synthesis for application B

- In order to switch from application A to application B it is necessary to stop the system and perform a complete reconfiguration of the FPGA



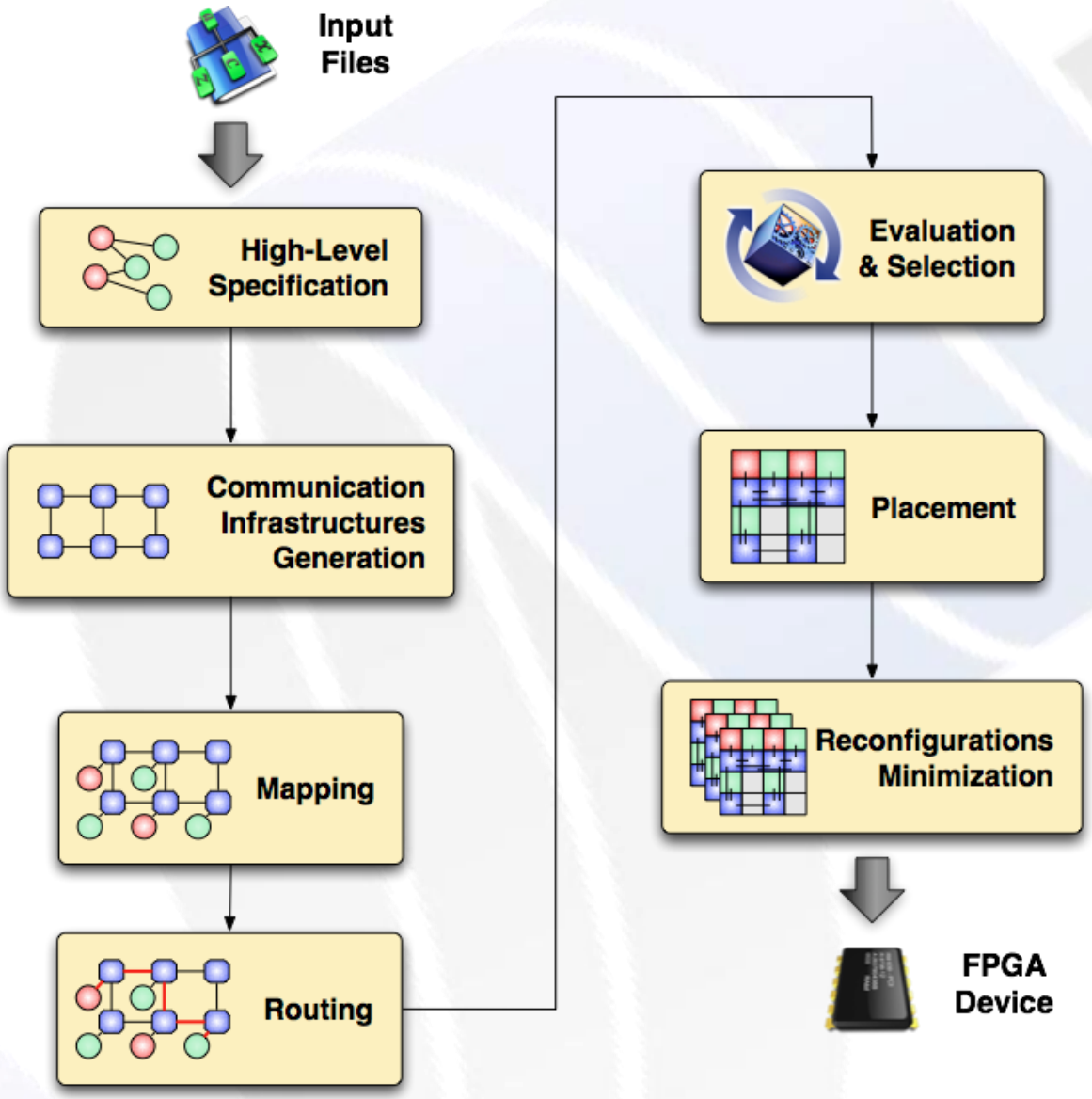
Possible approaches to reconfiguration



Design Framework goals

- Mapping and placement of applications
  - Multiple applications
- Minimization of reconfiguration overhead
  - Configuration time
  - Switching time of application contexts under tight timing constraints
  - Energy cost
- Maximization of the quality of the communication
  - Optimization of the mapping of the cores on the communication infrastructure for each application

Fine-Grained Design Flow

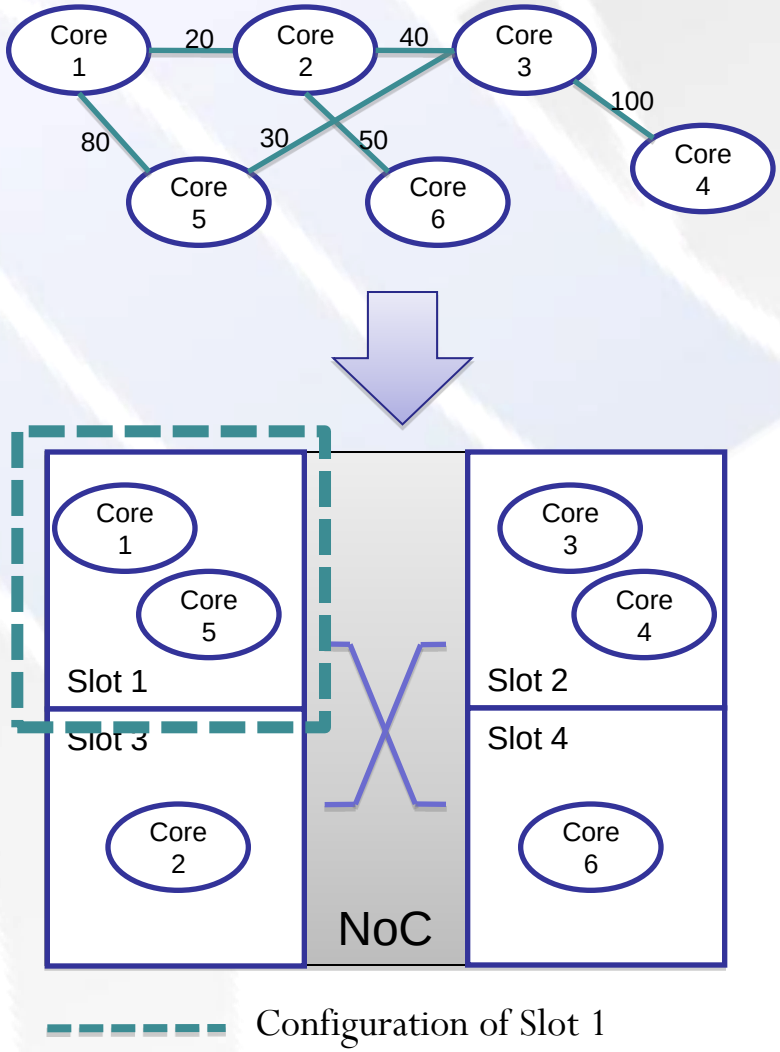


Fine-Grained Design Flow

- Communication Infrastructures Generation
  - ring, star, mesh, spidergon, custom
- Mapping and Routing
  - Exhaustive algorithm
    - Smart Exhaustive algorithm
  - Heuristic algorithms
    - Multi-objective genetic algorithm
      - NSGA2ver
    - Custom single-objective algorithms
      - GA1ver
      - GA2ver
- Effective only if all the cores are of the same size

Coarse-Grained Mapping Problem

- Mapping problem
  - Assignment of each core of each application to a slot
- Slot configuration
  - **List of cores** to be mapped into a specific slot
  - Encoded by a bitstream
  - May be reused
- Design-time mapper
  - Maps a known set of applications statically



Coarse-Grained Design Flow

- The proposed mapping flow consists of 3 stages, iterated until a feasible solution is found:
  - Preprocessing
    - Ordering
    - Selection (*beta*)
  - Partitioning
  - Mapping
    - Primary mapping
    - Secondary mapping (*alpha*)
- Convergence guaranteed by the *beta* parameter

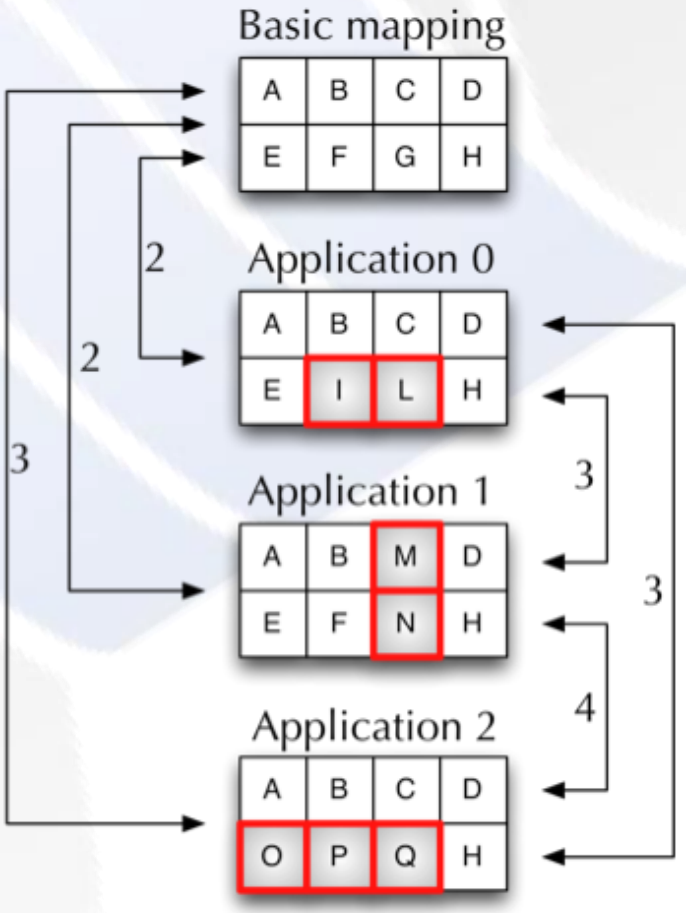
Preprocessing and Partitioning

- Preprocessing
  - Only a subset of all the cores needed by all the applications are firstly deployed on the device
    - **Ordering**: linear combination of the size of the cores and their utilization frequency
    - **Selection:** Only the biggest and most used cores are deployed on the device in this phase
  - Preprocessing makes it possible to exploit the similarities among the applications

- Partitioning
  - The subset of cores selected in the previous phase are partitioned by using the *Chaco* partitioner
  - The number of clusters (islands) has to be equal to the number of RRs of the target reconfigurable architecture

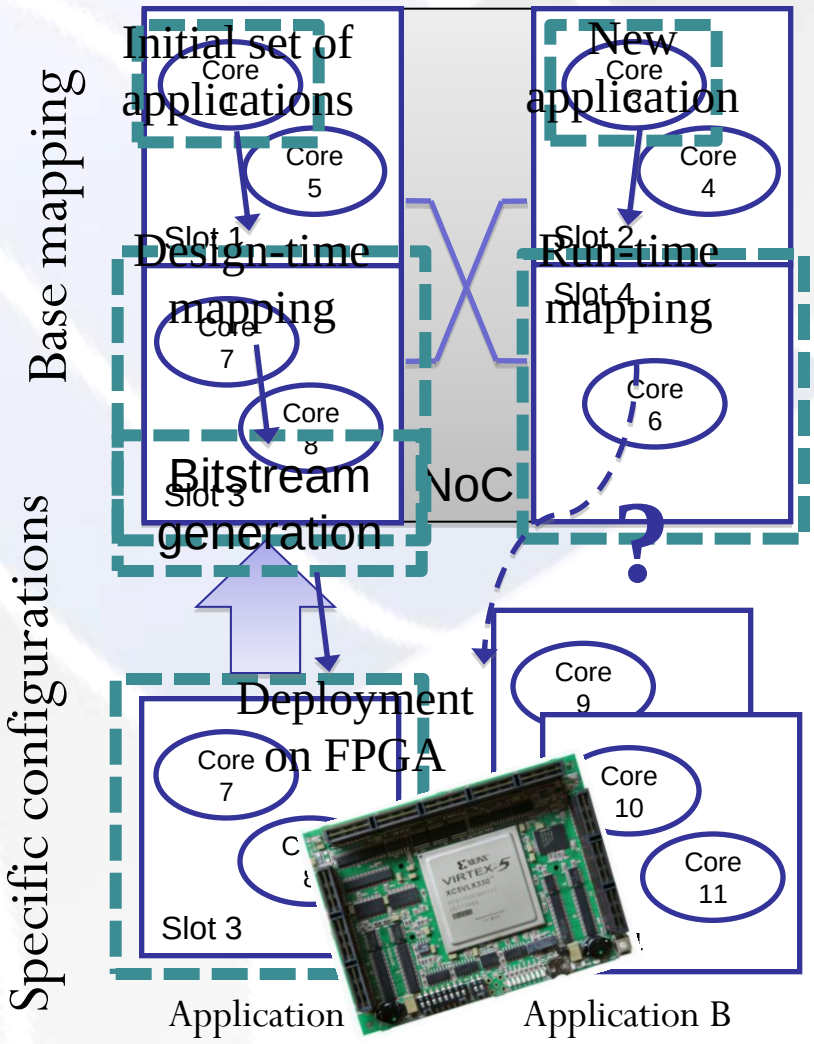
Mapping

- Mapping of the islands on the device
  - Primary mapping
    - The clusters obtained through the *Chaco* partitioner are mapped on the target architecture (one cluster for each RR) with a genetic algorithm
  - Secondary mapping
    - For each application, a subset of cores that are not useful for the currently selected application is removed
    - All the cores needed by the currently selected application are added to the system (creating new islands)



Toward a Run-Time Mapper (1/2)

- Structure of the solution
  - Base mapping
  - Specific configurations
- Run-time mapping
  - Mapping of a new application based on the existing base mapping
- Objective functions to be minimized
  - Average number of reconfigurations
  - Communication overhead
  - Number of new bitstreams to be generated



Toward a Run-Time Mapper (2/2)

- So far, we assumed that all the applications are known during the design phase
  - What happens if a new application is added later?
- The deployment time of the new application is related to the time we need to synthesize it
  - The number of bitstreams to be generated should be low
  - The base mapping should not change
- A design-time algorithm cannot be used
  - Rather complex (partitioner, genetic algorithm, …)
  - It may generate a different base mapping

Run-Time Mapper: Overview

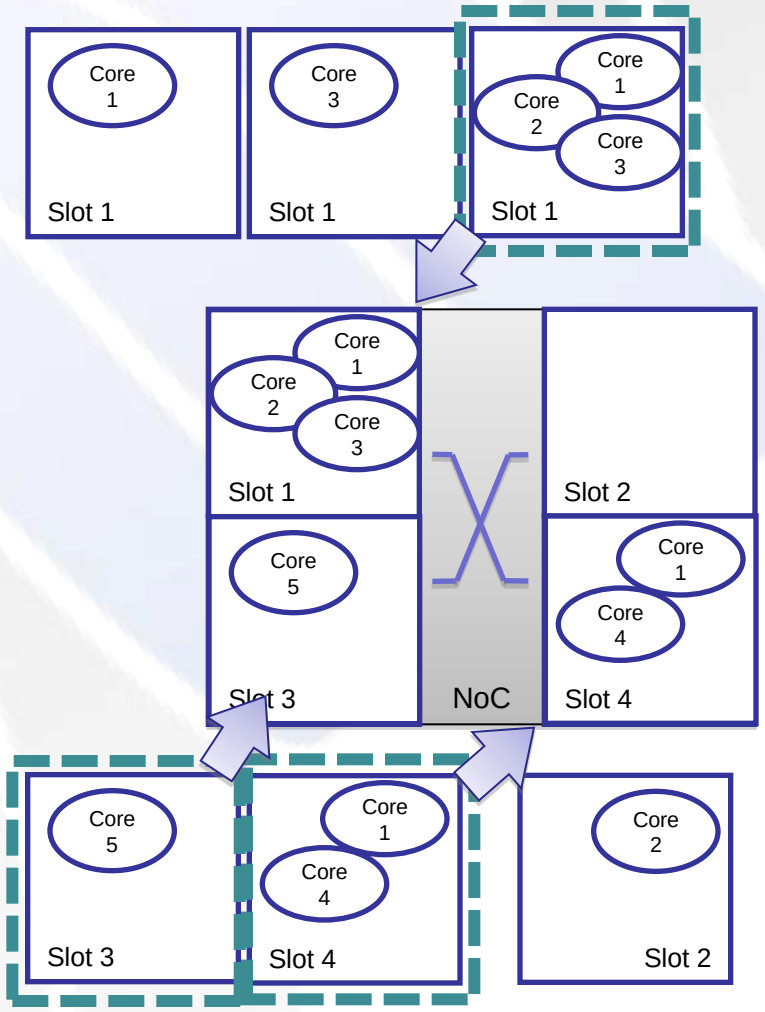
- Idea: try to deploy **at least** part of the new application by reusing the existing configurations

- If the incoming application does not introduce any new core we may map it using existing configurations only
  - No new bitstream is generated
  - The proposed approach makes it possible to immediately deploy the application

  - Finding a feasible solution is an intractable problem
    - Proved to be *NP-Complete*
    - For n application and m slots, there are up to n^m combinations

Run-Time Mapper: Algorithm

- We propose a fast heuristic technique
  - It can quickly find a solution, or declare that it does not exist
  - It tries to optimize the objective functions, whenever possible
- The configurations are iteratively included in the solution
  - A score is computed at every iteration for each configuration



Run-Time Mapper: Algorithm

- Guidelines to compute the score
  - Two different configurations of the same slot are mutually exclusive
  - Configurations containing several cores are preferred because they do not waste area
- Basic formula to compute the score of a configuration *i*:

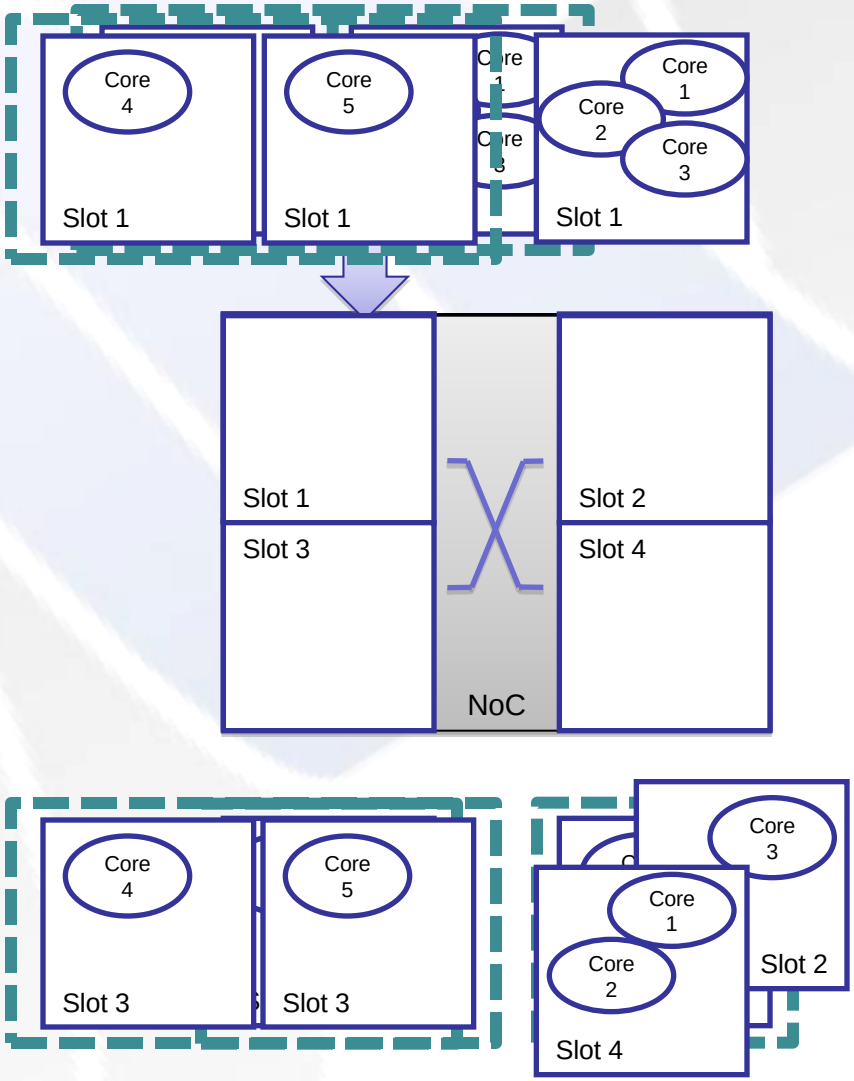
$$Score_i = \begin{cases} 0 & \text{if the slot is already used} \\ \alpha \dfrac{Useful\_Area}{Slot\_Area} + \beta \dfrac{Internal\_Communication}{Total\_Communication} & otherwise \end{cases}$$

Affects the average number of reconfigurations

Affects the communication overhead

Run-Time Mapper: Common Traps

- The mapper may be trapped when single instances are not selected
  - In practice they are frequent
  - The score of single instances is forced to a very high value
- Single instance chains
  - The detection becomes very complex
  - In practice they are rare
  - They are not considered by the algorithm



General Case RT-Mapper (1/3)

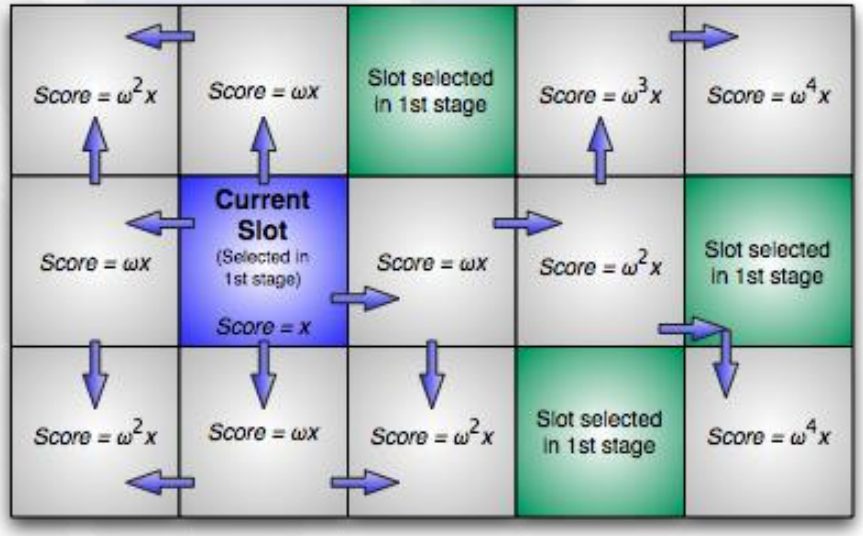
- If at least one core was not known at design time, bitstream generation is unavoidable
  - We can reduce the number of new bitstreams, and hence the deployment time…
  - … While still working on the number of reconfigurations and the communication overhead
- The algorithm is divided into three stages:
  - Configuration reuse
  - Core sorting
  - Mapping

General Case RT-Mapper (2/3)

- Configuration reuse is still an option, but it is not enough to build a complete solution
  - Existing configurations may even waste area and affect the feasibility of the solution
- Keep on selecting configurations until a termination condition is met
  - Pick configurations that do not waste area and resolve communication internally
  - Stop when the area on the device becomes low
- Sort the remaining cores according to their criticality
  - Larger cores are more critical
  - Cores generating a high communication are also critical

General Case RT-Mapper (3/3)

- For each remaining core, the best slot is selected by means of a propagation technique
  - The communication between the core and the already-mapped ones is computed
  - The communication value associated with each slot is propagated over the mesh
  - The value of partially-occupied slots is increased to reduce the number of bitstreams
  - The slot with the highest value is picked for the new core



$\omega < 1$

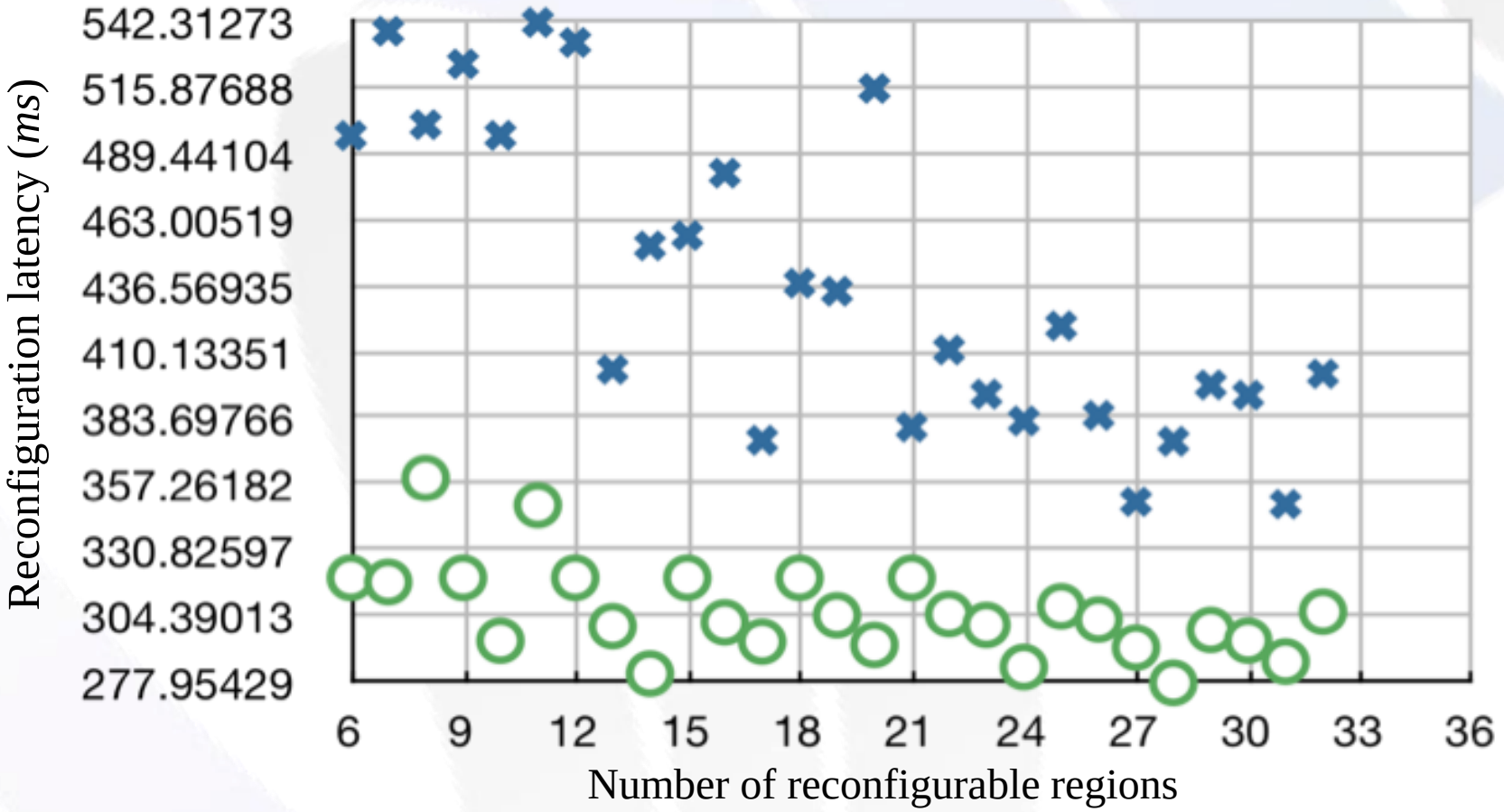
Design Frameworks comparison

- Comparison among state-of-the-art approaches and the proposed Design Framework

	[58]	[59]	[61]	[69]	[62]	[68]	[70]	[71]	[55]	[72]	[73]	[74]	Proposed Design Flow
Support for ASIC design	No	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No	No	Yes	No
Rec. Support	No	No	No	No	No	No	No	Yes	Yes	Yes	Yes	No	Yes
Cores Rec.	No	No	No	No	No	No	No	Yes	Yes	No	No	No	Yes
Non-homogeneous Tiles	No	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
NoC topology Rec.	No	No	No	No	No	No	No	No	Yes	Yes	No	No	Yes
Standard and Custom Topologies	Yes	No	No	Yes	Yes	No	Yes	No	Yes	Yes	No	No	Yes
Mapping Phase	Yes	Yes	Yes	Yes	Yes	No	Yes	No	No	No	No	No	Yes
Routing Phase	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	No	No	No	Yes
Placement Phase	No	No	Yes	Yes	No	Yes	Yes	No	No	No	No	No	Yes
From HLS to bitstreams	No	No	No	No	No	No	No	No	No	No	Yes	Yes	Yes

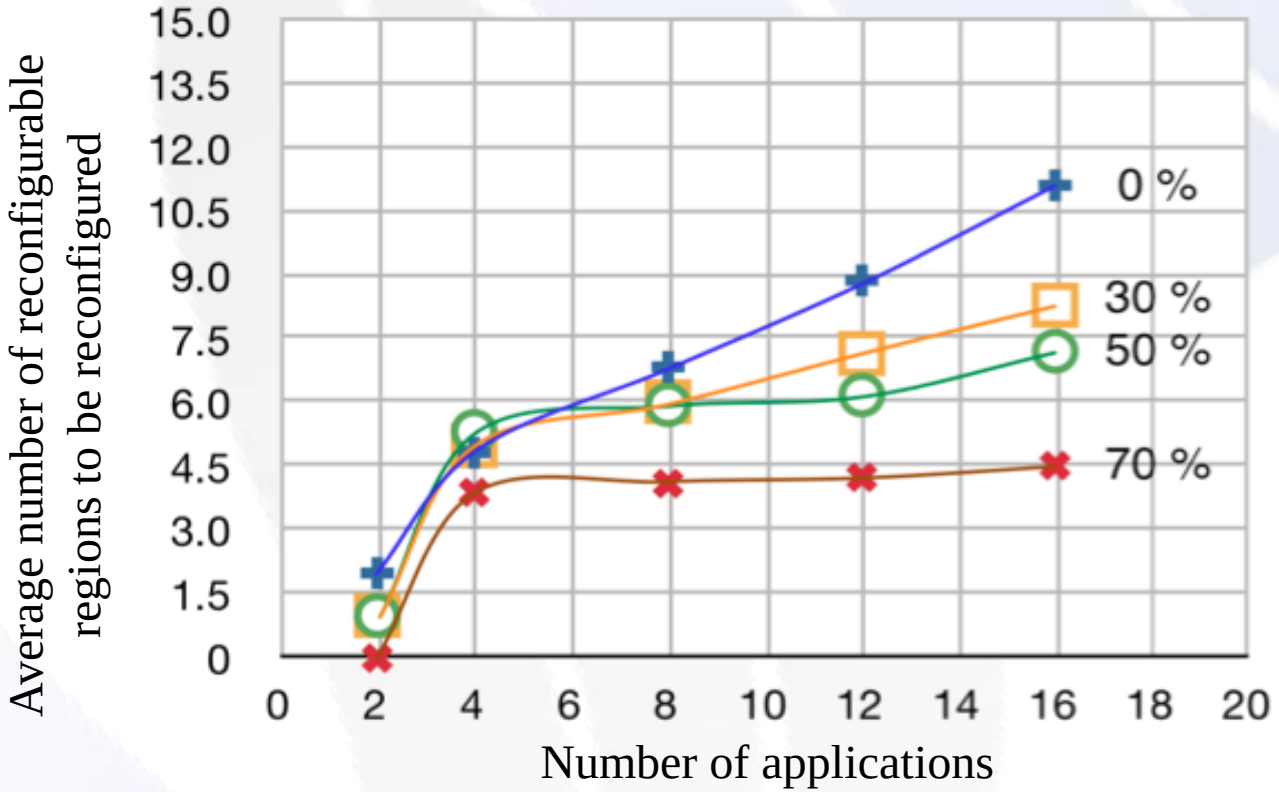
Experimental Results (1/2)

- Evaluation of the Coarse-Grained Design Flow
- Proposed approach (o) and an approach that does not consider reconfiguration costs (x)



Experimental Results (2/2)

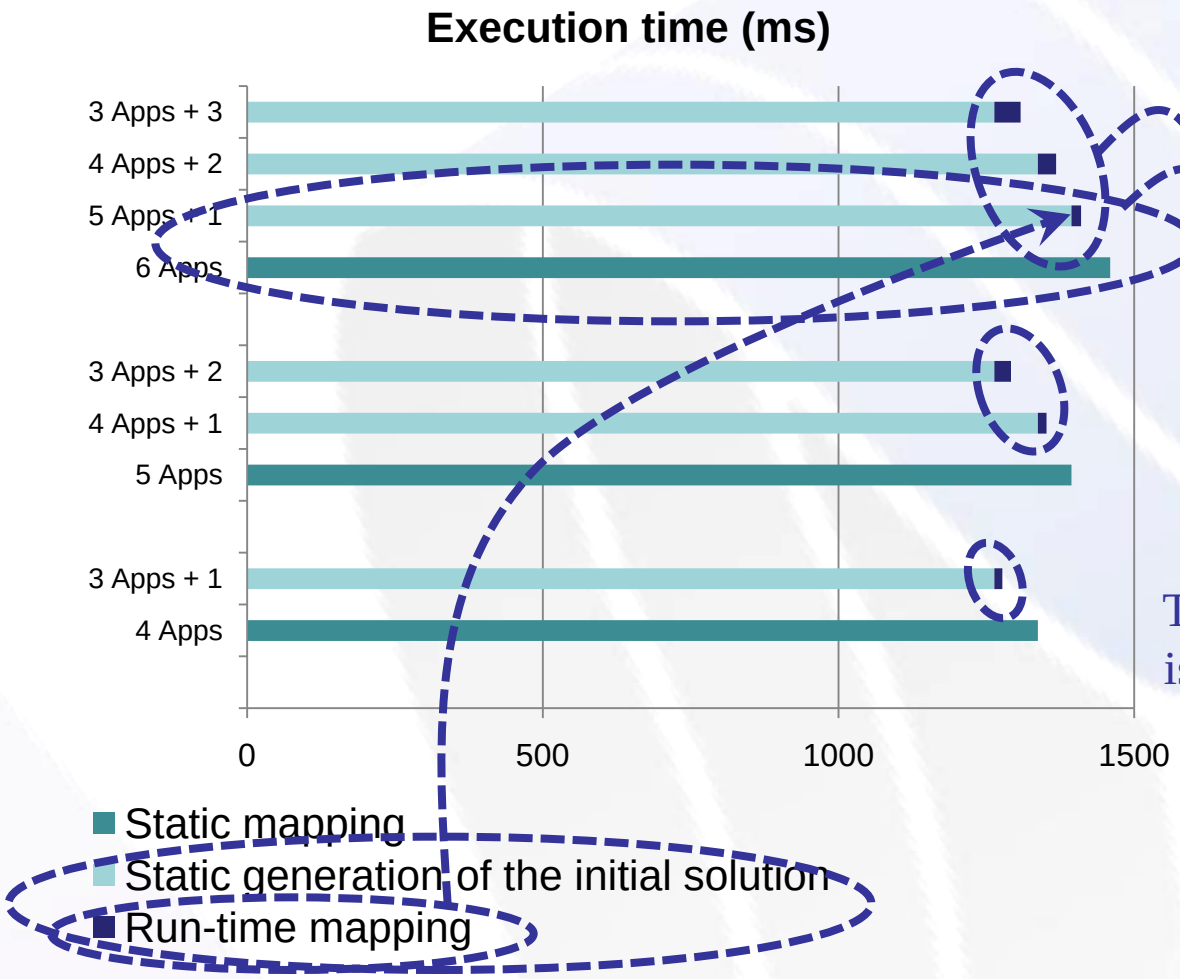
- Evaluation of the Coarse-Grained Design Flow
- The percentage of shared cores is set to 0%, 30%, 50% or 70%
- The proposed approach scales very well when the number of applications increases, if they are characterized by enough similarities



Experimental Setup

- Multiple sets (of *n* applications each) mapped statically
  - Unless different specified, *n* = 5
  - Each application needs between 10 and 35 cores
  - Approximately 70% of shared cores
- Additional *m* applications are added at run-time
  - *m* ranges from 1 to 3, depending on the test
  - All the cores are included in at least one of the *n* applications mapped statically
- 16-slots hardware architecture
  - Target device: Xilinx Virtex-4 (Xilinx XC4VLX40)

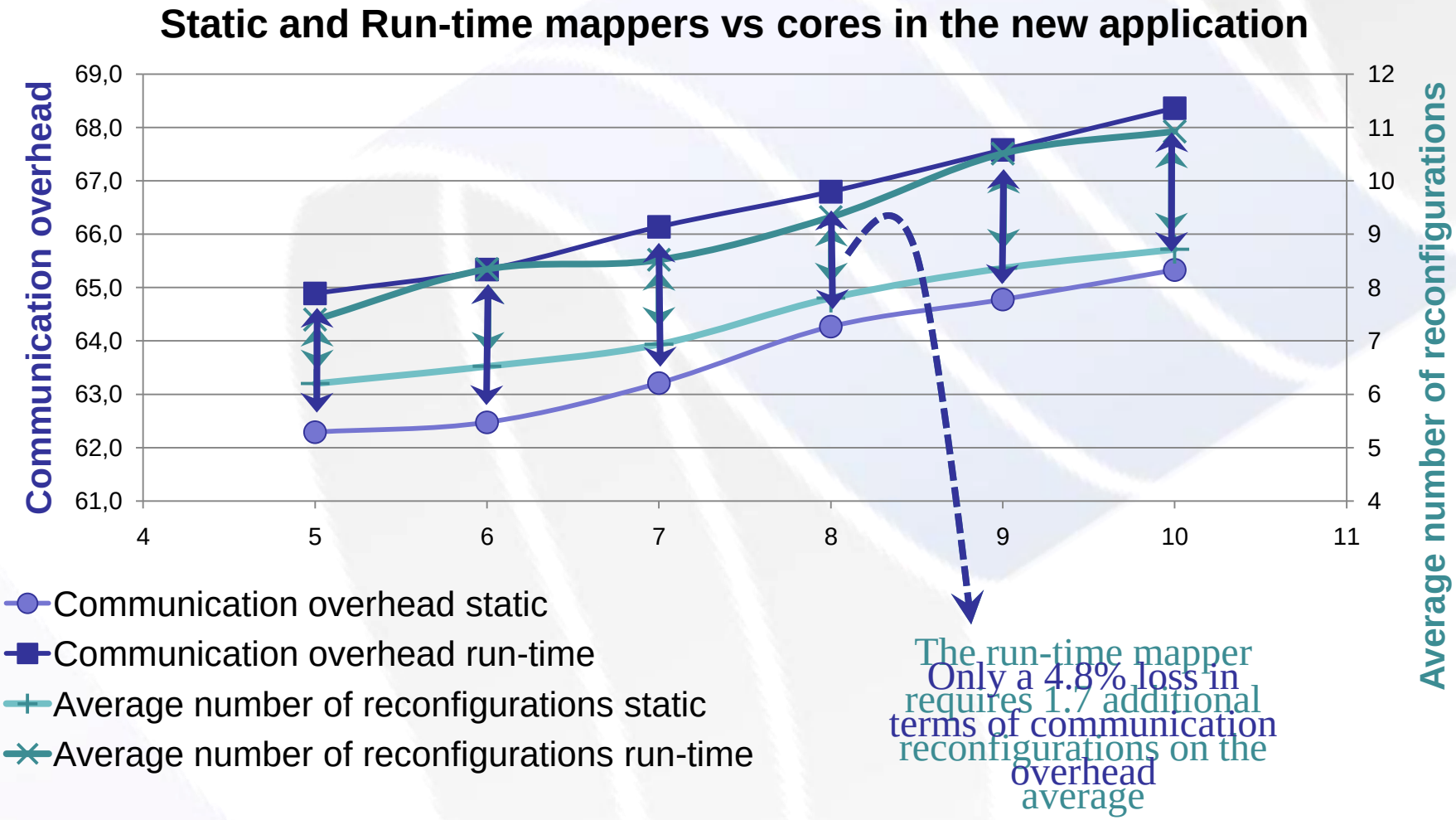
Execution Time



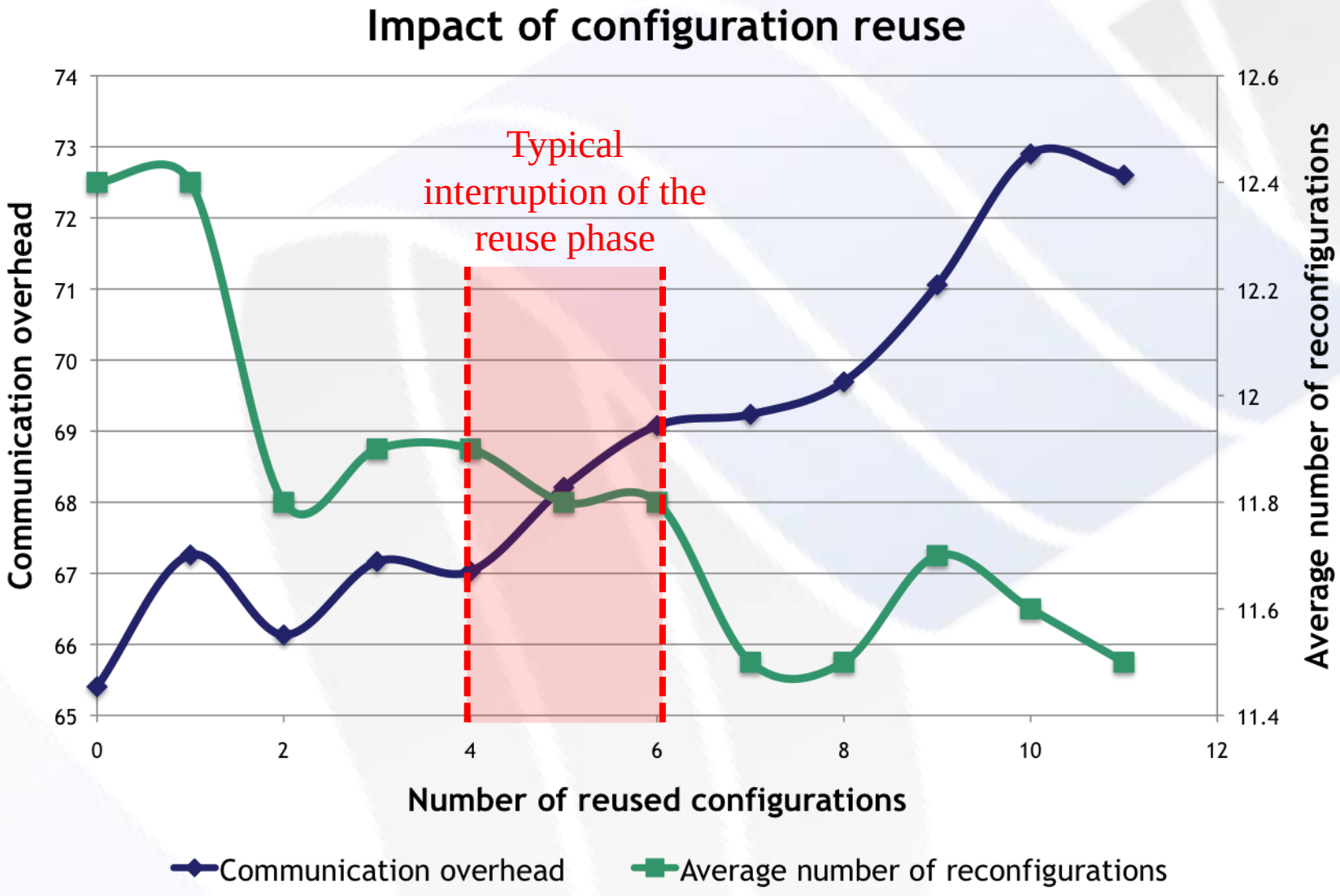
At least 26.5 times faster than an execution of the static mapper

The combined execution time is faster than a new run of the static mapper

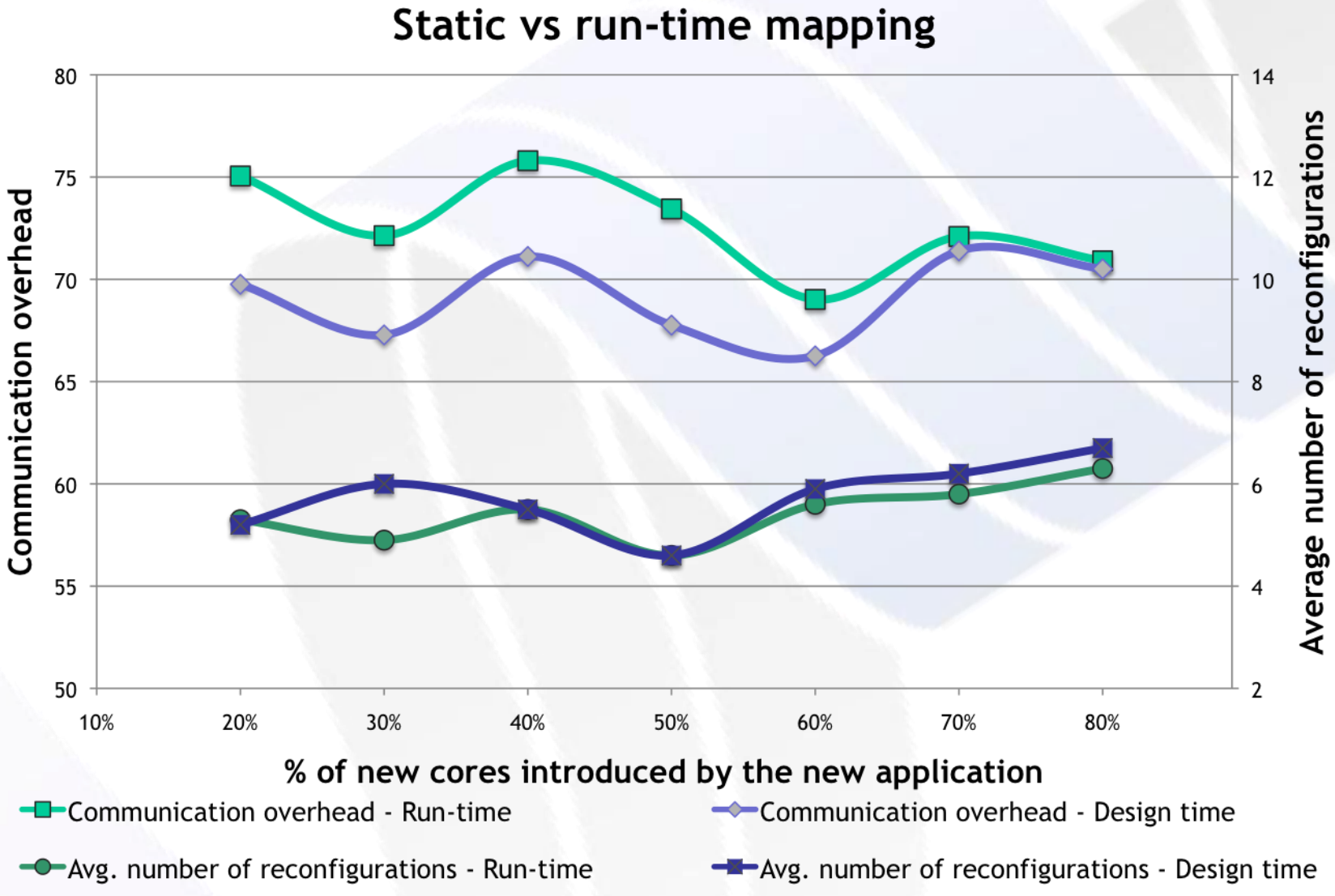
Evaluation Of Configuration Reuse



General Case RT-Mapper Results (1/2)



General Case RT-Mapper Results (2/2)



Concluding remarks (1/2)

- The proposed NoC
  - is able to completely support reconfiguration
  - is characterized by a very low latency
- The proposed design framework
  - is able to automatically generate a reconfigurable embedded system minimizing
  - the communication overhead
  - the reconfiguration overhead
  - consists of several algorithms characterized by very good timing performance

Concluding remarks (2/2)

- Mapping of multi-core applications on reconfigurable devices
  - Switching between applications at run-time
- Complete framework to solve the mapping problem
  - A design-time algorithm to handle a known set of applications
  - A run-time algorithm to deploy a new application at any time without re-synthesizing all the applications in the system
- The application is deployed quickly with a reasonable quality

Questions

Thanks for your attention!

Any question?