Multicore Acceleration of System-level Simulation

Salah Eddine SAIDI, Abir BEN KHALED,
Yves SOREL, Nicolas PERNET, Mongi BEN GAID

Outline

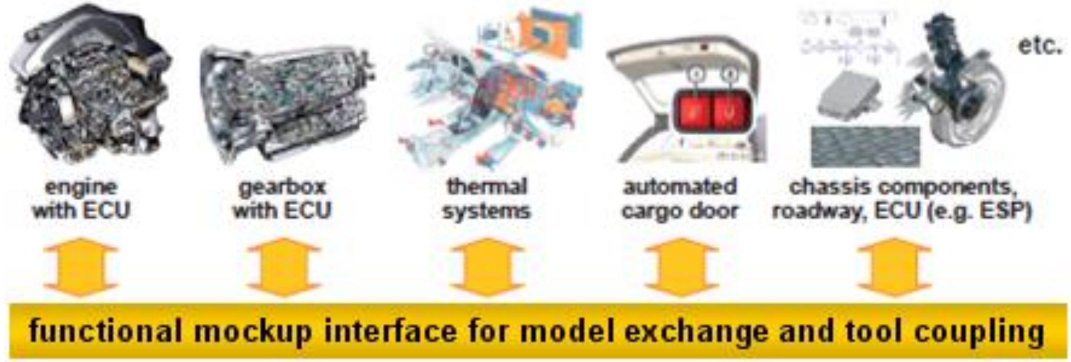
- Context
- Problem description
- IFPEN results on simulation acceleration
  - Splitting is speed-up
  - Ensuring speed-up and accuracy: the RCOSIM method
  - Context-based extrapolation
  - Mapping real-time constraints on a system-level simulation
- Future work

Outline

- Context
- Problem description
- IFPEN results on simulation acceleration
  - Splitting is speed-up
  - Ensuring speed-up and accuracy: the RCOSIM method
  - Context-based extrapolation
  - Mapping real-time constraints on a system-level simulation
- Future work

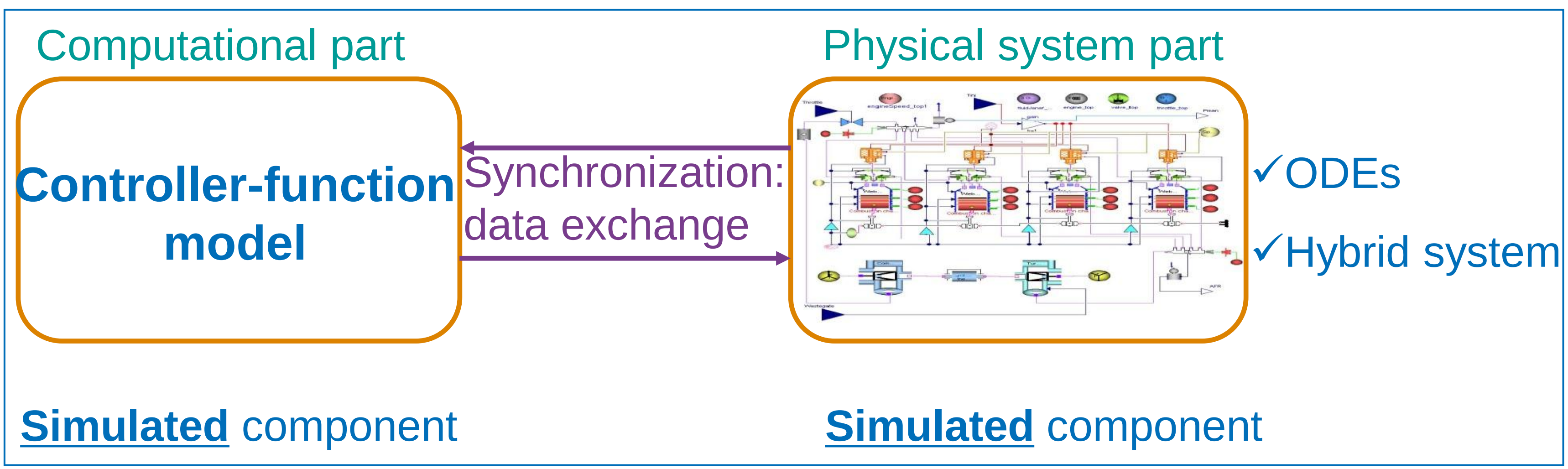
Multi-model integration for system-level simulation

- TODAY:
  - Simulation = a key factor for development cost reduction
  - 0D models = the good modeling level for collaborative development
  - Different domains = different modeling tools



Simulation approaches

- System Simulation (0D) → ODEs
- Co-simulation
  - Heterogeneous models: different domains, different tools
  - Synchronization between models, Calculations ASAP
  - → Prototyping and validation



Simulation approaches

- System Simulation (0D) → ODEs
- Hardware-in-the-Loop simulation
  - Real components (ECUs) + simulated models (engine, powertrain)
  - RT constraints
    - Execution rate, components synchronization
    - Computation times ≤ RT deadlines

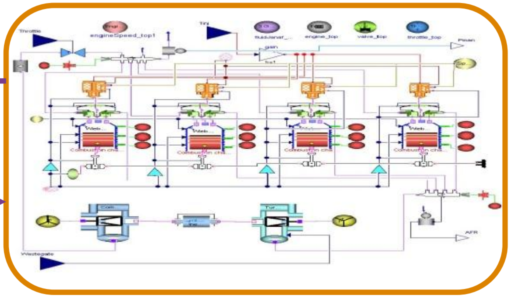
Computational part

Engine Control: ECU
Real component

Synchronization: data exchange

Real-Time

Physical system part



Engine Simulator
Simulated component

✓ODEs
✓Hybrid system

A multicore simulation Kernel: Why?

- System-level simualtion leads to agglomerate models which are classically disconnected, increasing the CPU demand at simulation time
- Simulation time becomes more and more a metric for model complexity
- Most 0D/1D simulation tools have monocore kernel while monocore computers are endangered

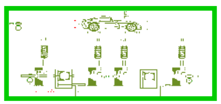
➔ How much more will this CPU power remain unused ?

Outline

- Context
- Problem description
- IFPEN results on simulation acceleration
  - Splitting is speed-up
  - Ensuring speed-up and accuracy: the RCOSIM method
  - Context-based extrapolation
  - Mapping real-time constraints on a system-level simulation
- Future work

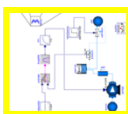
Problem description:
Acceleration of multi-model simulation

AMESim®



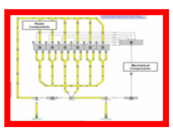
Vehicle

Dymola®



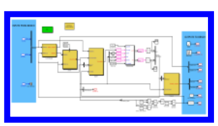
Driver

GT-Power®

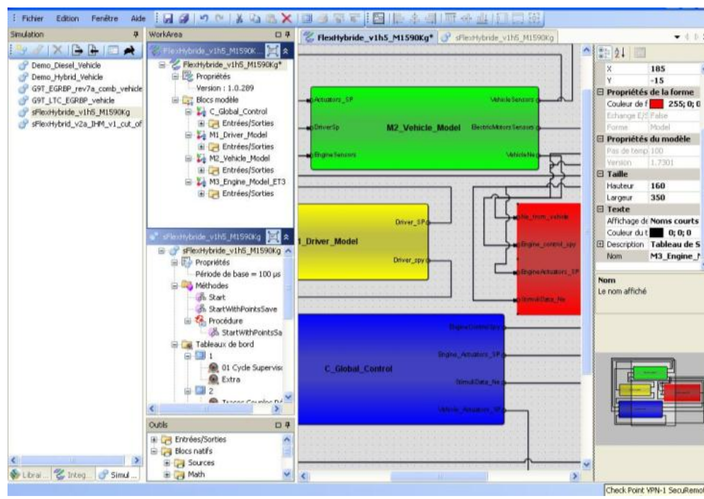


Engine

Simulink®



Control



ASAP simulation

→ **Acceleration by multicore**

Real-time simulation for HiL

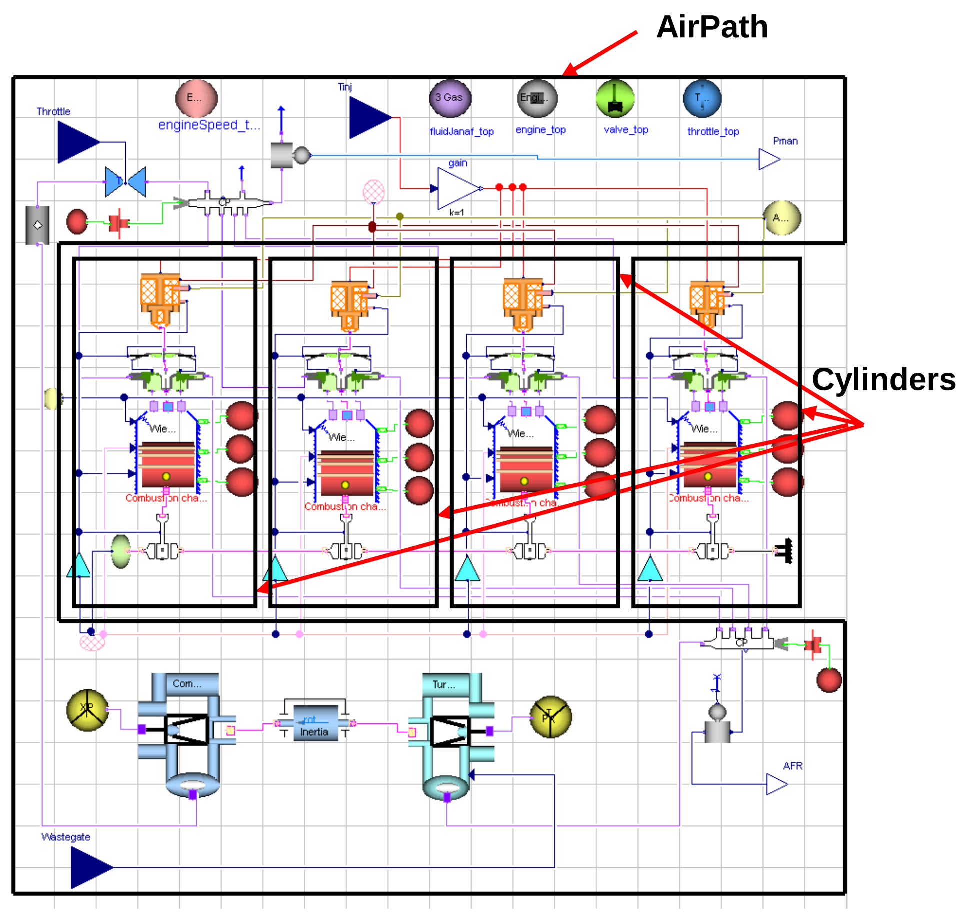
→ **Acceleration by multicore**
+
Real-time constraints propagation

Outline

- Context
- Problem description
- IFPEN results on simulation acceleration
  - Splitting is speed-up
  - Ensuring speed-up and accuracy: the RCOSIM method
  - Context-based extrapolation
  - Mapping real-time constraints on a system-level simulation
- Future work

Model splitting from a physical point of view

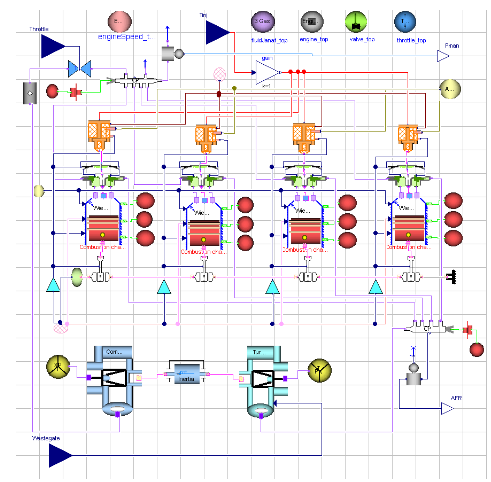
- **Remark**
  - Events are related usually to the evolution of a subset of the state vector
  - Discontinuities are independent from a physical point of view
- **Partitioning engine model**
  - ↓ Discontinuities (locally)
  - → Improve efficiency ?

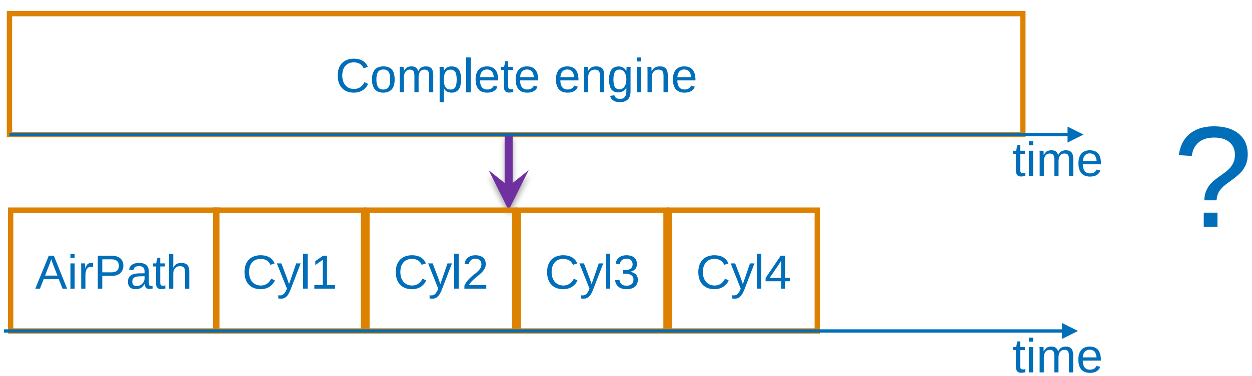


Model splitting from a physical point of view
Effect on simulation time (single-core)

- Test case: Wiebe model + LSODAR solver
- Comparison of simulation time
  - Single-thread single-core approach: original model
  - Multi-threads single-core approach: split model

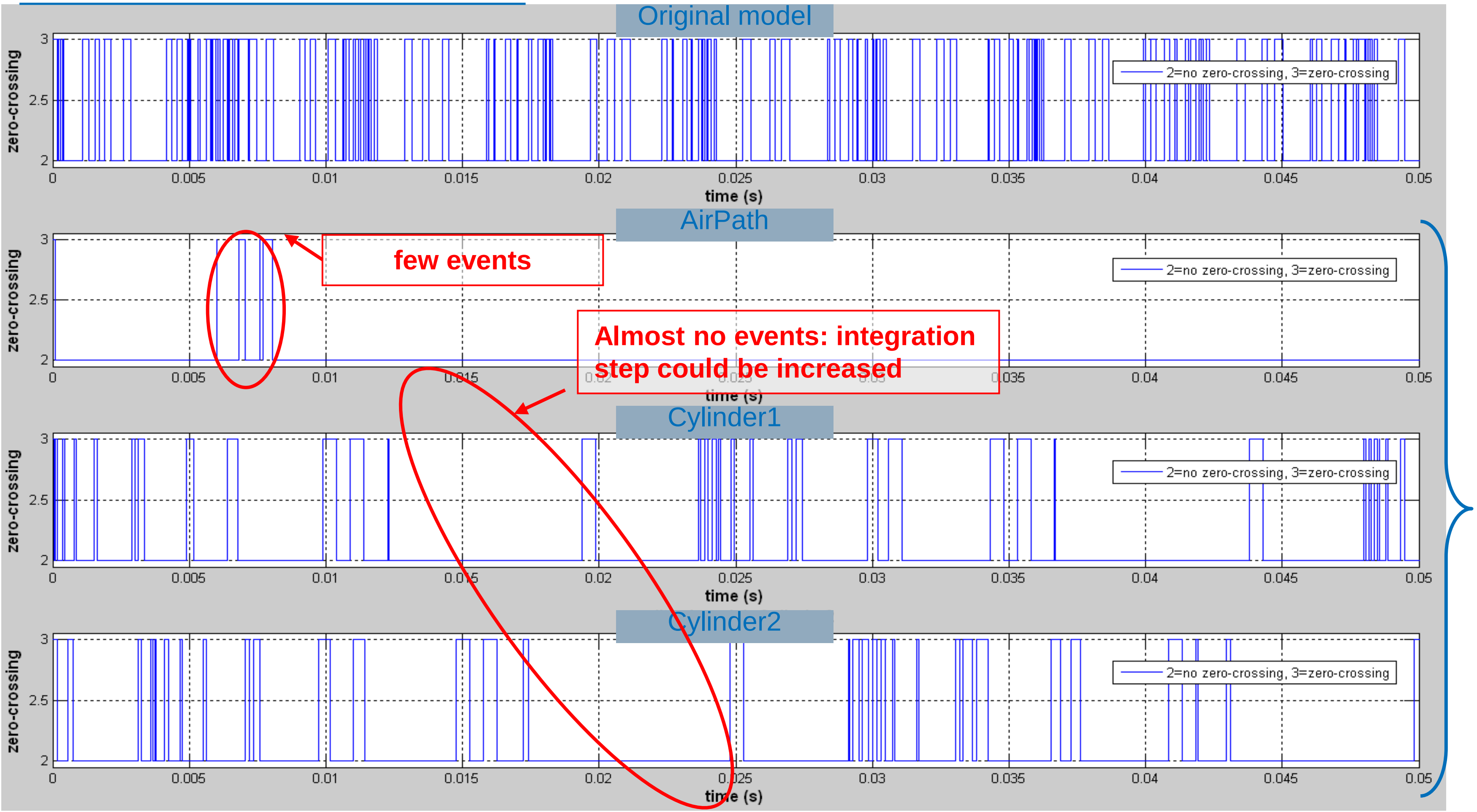
    →See only the effect of events relaxation on the speed-up of LSODAR solver without the effect of the parallelization





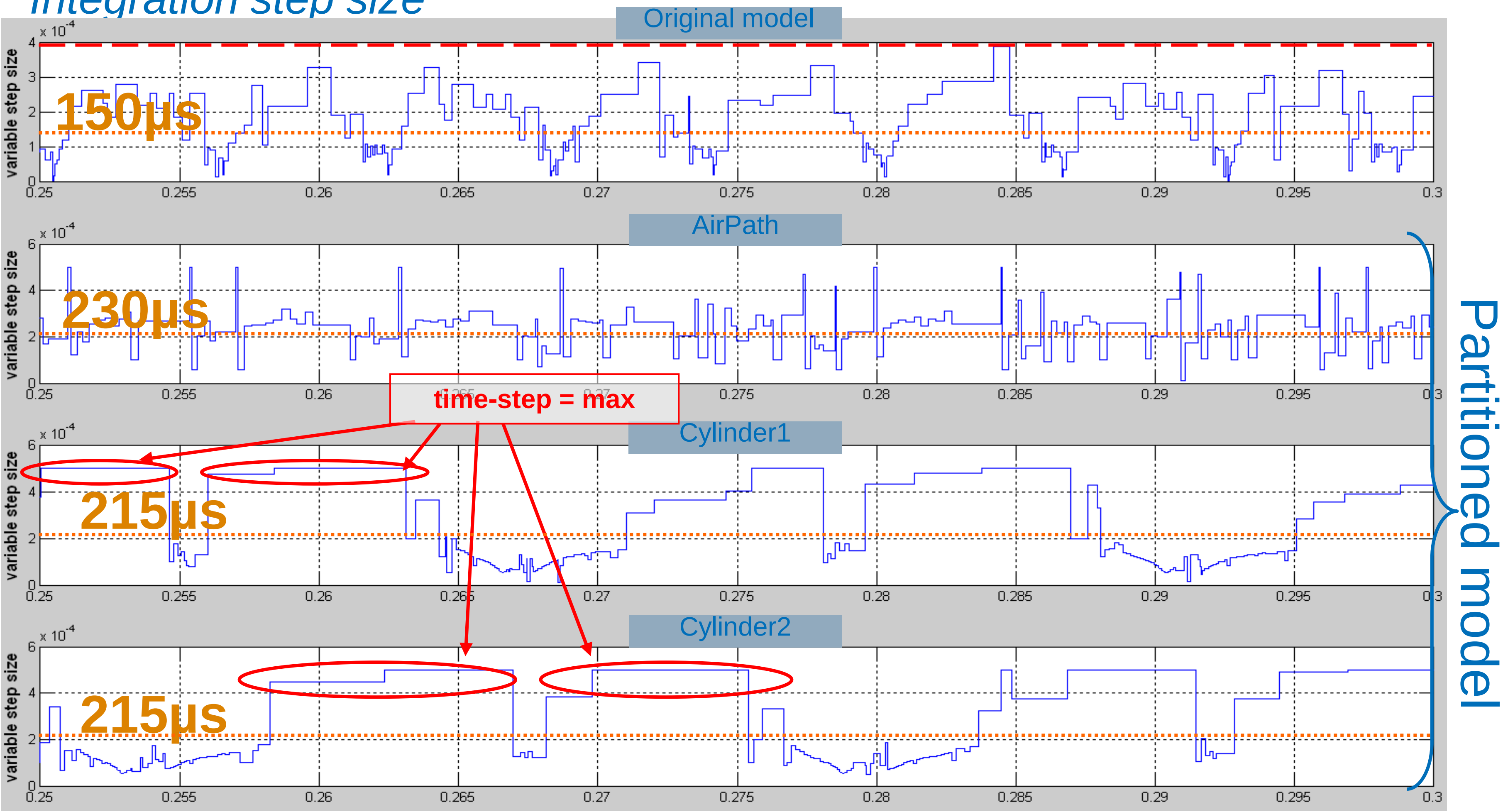
Model splitting from a physical point of view Effect on simulation time (single-core)

Number of discontinuities



Model splitting from a physical point of view
Effect on simulation time (single-core)

Integration step size



Model splitting from a physical point of view
Effect on simulation time (single-core)

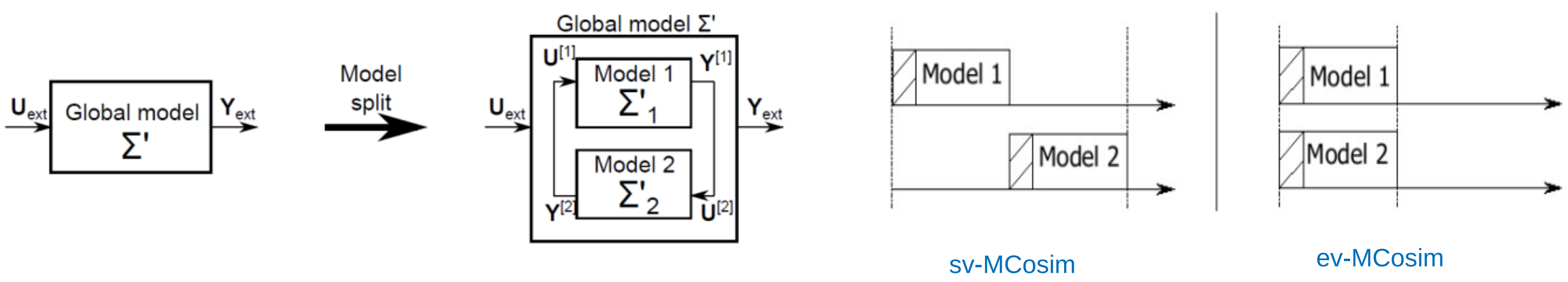
- **The execution of the split model is almost <u>twice</u> faster than the original model**
  - ➔ Speed-up = <u>1.98</u>
  - ➔ Thanks to the system decomposition
    the use of a single solver per sub-system
  - Despite multi threading cost
  - The parallelism effect is not yet taken into account

Outline

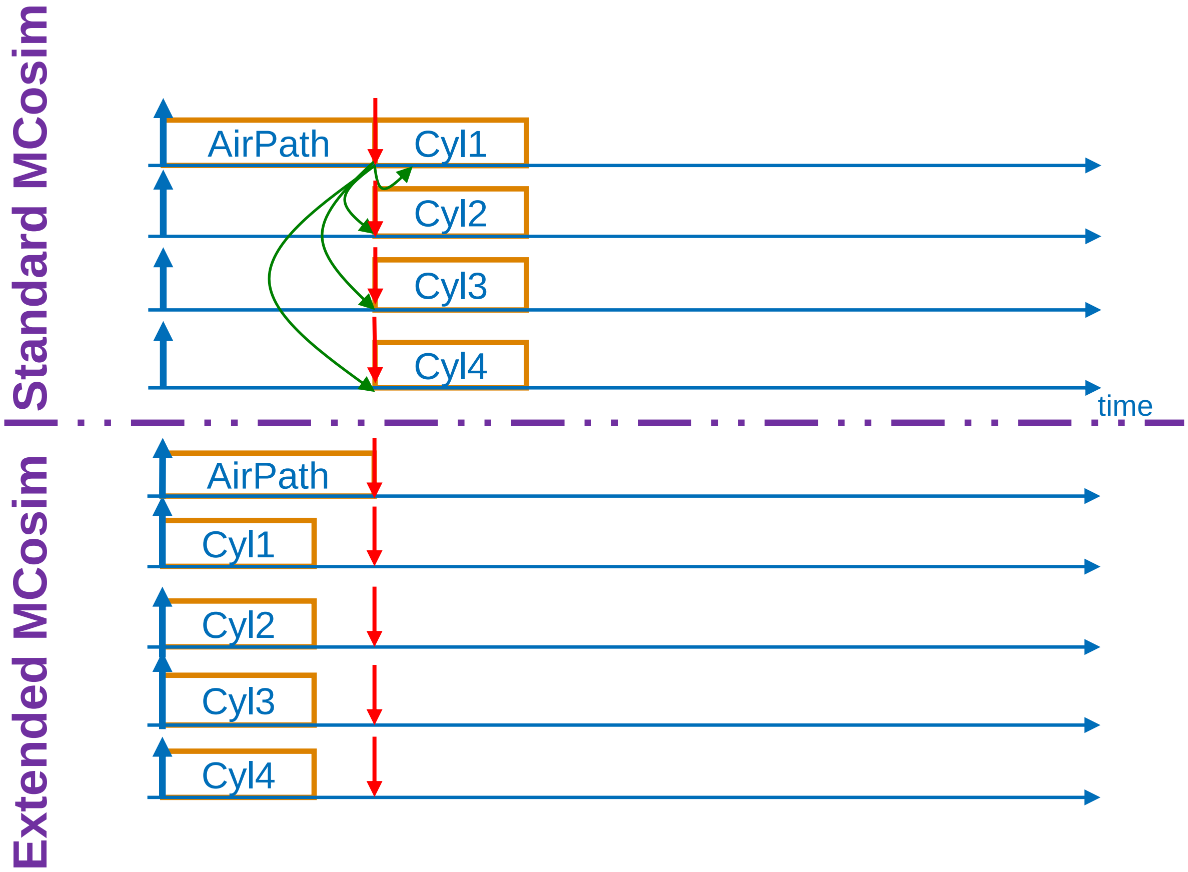
- Context
- Problem description
- IFPEN results on simulation acceleration
  - Splitting is speed-up
  - Ensuring speed-up and accuracy: the RCOSIM method
  - Context-based extrapolation
  - Mapping real-time constraints on a system-level simulation
- Future work

Multi model execution on multicore

- Partitioning process ➔ generation of loops
- After partitioning a model: execution order?
  - The standard version: sv-MCosim
    - Most of data dependencies are respected
  - The extended version: ev-MCosim
    - Data dependency constraints are relaxed to achieve a better speedup



Model splitting from a physical point of view
sv-MCosim and ev-MCosim (multi-core)

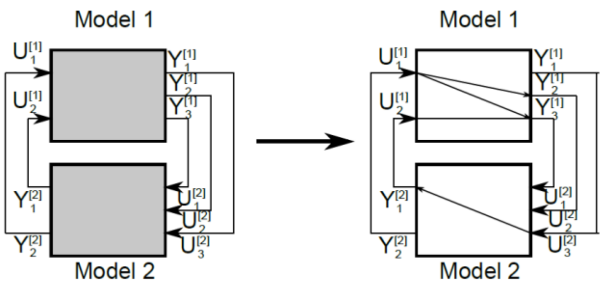


Dependencies respected
→ Speed up = 3,15
→ - Accuracy (Delays)

Dependendencies removed
→ Speed up = 3,9
→ -- Accuracy

RCosim: Refined scheduling Co-simulation

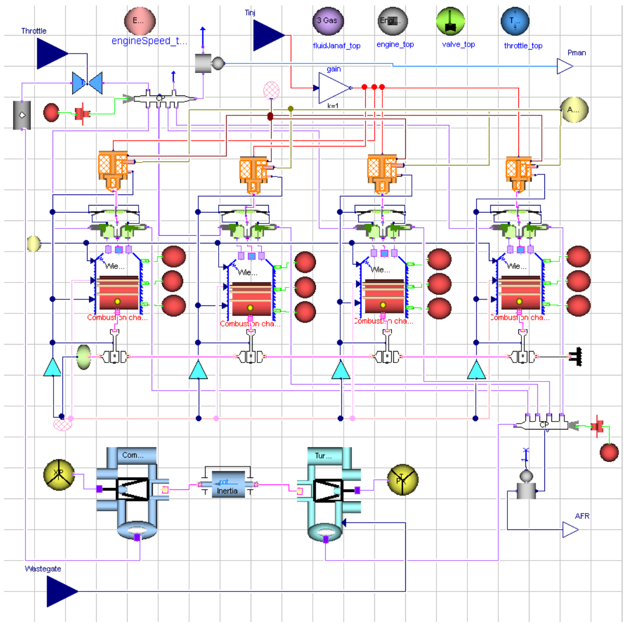
- RCosim: identify locally if Y is dependent on U or not?
  - FMI gives relationships between each Y and U
  - With FMI each I/O is computed with a different operation
- Off-line heuristic approach
  - Similar to SynDEx (INRIA) [Grandpierre and Sorel, 2003]
- Objective: Minimize critical path (CP) latency of DAG



Rcosim approach

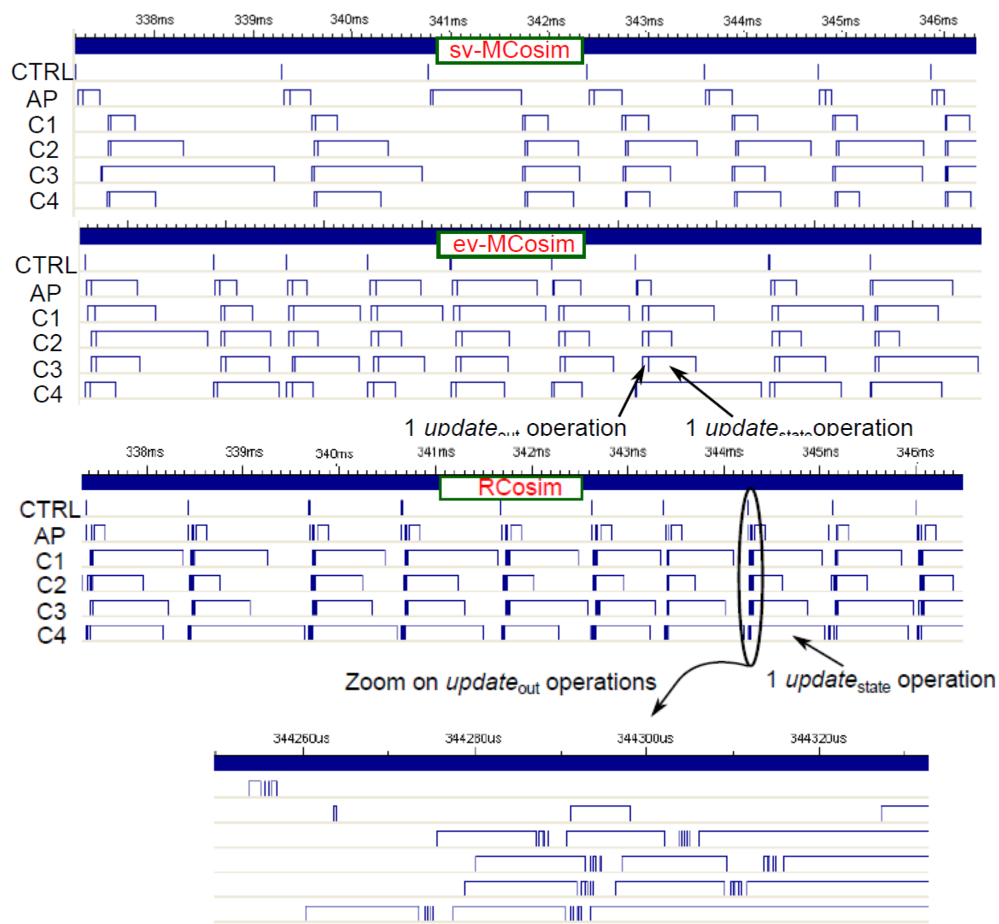
Case study: 4-cylinder internal combustion engine

- Engine model: Spark Ignition F4RT engine (Renault)
  - 4 cylinders + Air Path (turbocharger, throttle, wastegate,…)
  - 118 states
    - e.g. crank shaft angle, mass of gas, energy, temperature,…
  - 398 event indicators
    - e.g. spark advance time, engine cycle, intake valve lift,…
    - Trigger events, mathematical exception handling
  - 103 operations (update$_{out}$ …)
- Modeling & simulation tools
  - Dymola with ModEngine library + xMOD with FMU



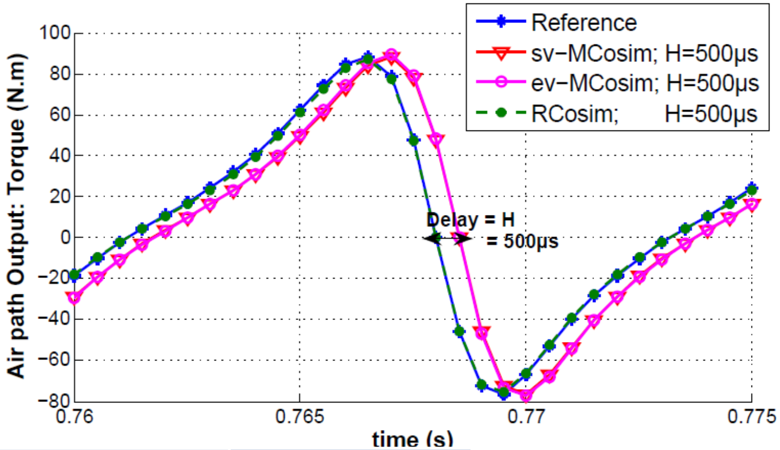
RCosim approach
Scheduling of update operations with RCosim

- **Reminder of the different models of computation**
  - sv-MCosim, ev-MCosim, RCosim
- **RCosim**
  - 103 operations
    - $Update_{all\_out}$ and $update_{all\_state}$
  - C(update_all_out) ≪ C(update_all_state)



RCosim approach
Simulation accuracy improved

- Torque is Direct-Feedthrough (DF)
  - numerical error (delays) with MCosim
  - no delays with Rcosim



Simulation method	sv-MCosim	ev-MCosim	RCosim
Er(%) with H=100µs	2.95	4.38	0.68
Er(%) with H=250µs	9.12	9.33	1.1
Er(%) with H=500µs	19.83	19.19	1.37

Simulation method	sv-Mcosim	ev-MCosim	RCosim
Er(%) with H=100µs	0.61	0.63	0.5
Er(%) with H=250µs	1.2	1.11	0.88
Er(%) with H=500µs	1.8	1.75	1.23

Rcosim approach
Simulation speed-up

Simulation method	sv-MCosim	ev-MCosim	RCosim
Speed-up (5 cores) Compared to a single-threaded single solver ref.	7,82	8,84	10,87
Speed-up (5 cores) Compared to a split model on single core.	3,94	4,64	5,48

- Speed-up > **5** ➔ supra-linear
- RCosim even faster than ev-MCosim
  - thanks to the variable step solver (less iterations)
- With a fixed-step solver :
  - Speed up close to ev-cosim, better than sv-Cosim
  - No broken cycle ➔ results are strictly identical to single model / single solver simulation

Outline

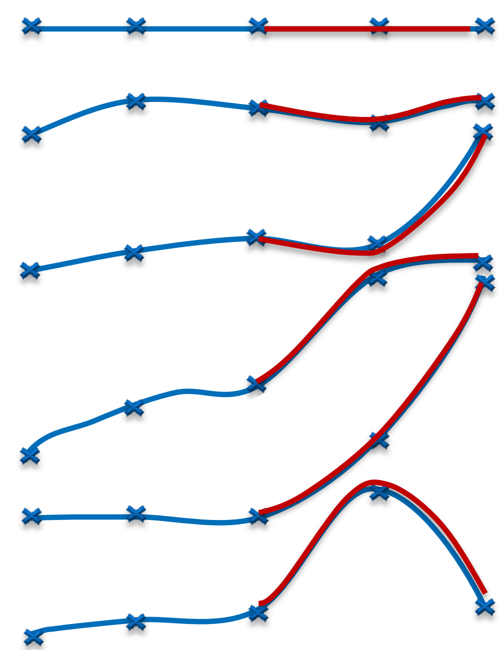
- Context
- Problem description
- IFPEN results on simulation acceleration
  - Splitting is speed-up
  - Ensuring speed-up and accuracy: the RCOSIM method
  - Context-based extrapolation
  - Mapping real-time constraints on a system-level simulation
- Future work

Context-based extrapolation
Difficulties and challenges

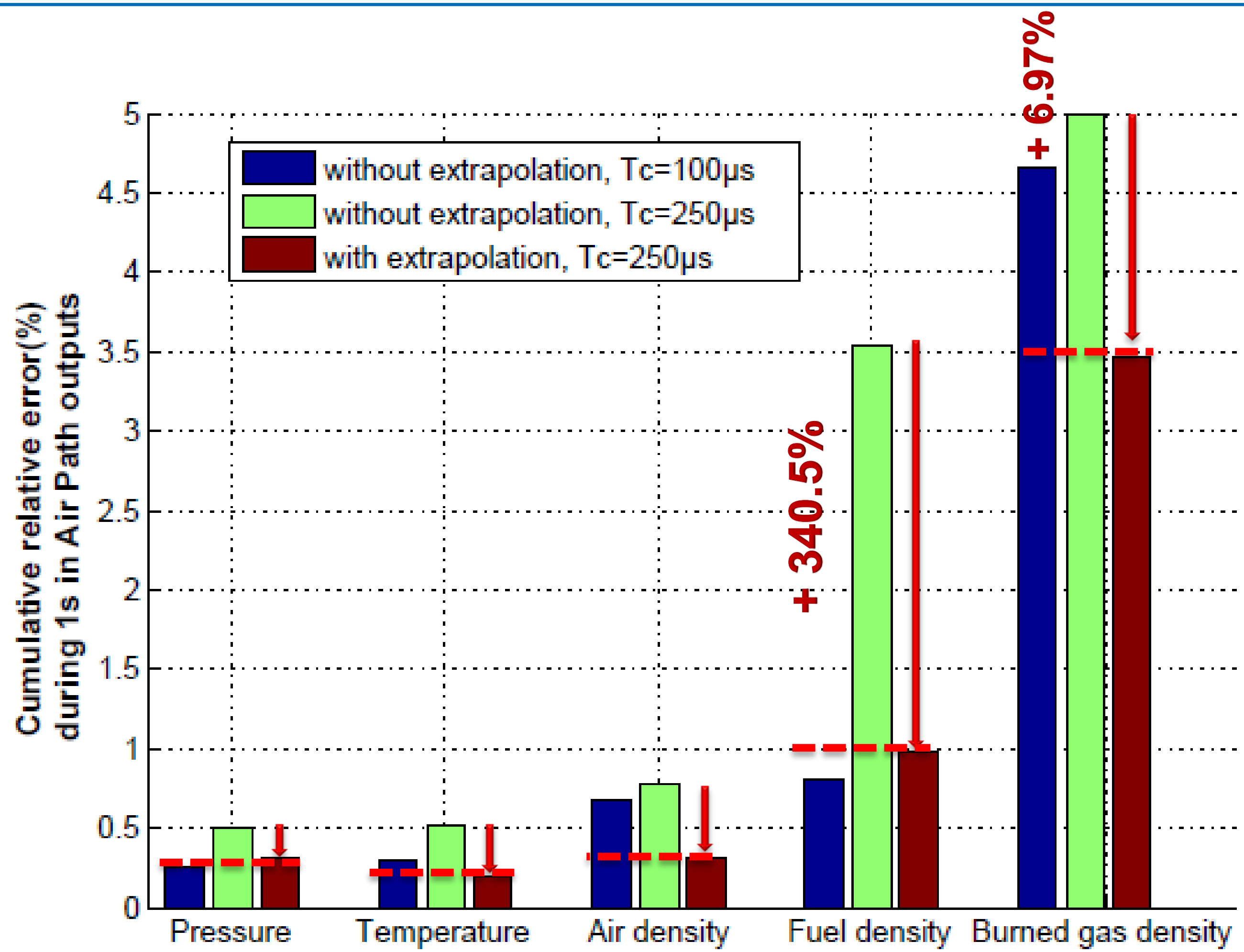
- Hybrid behaviors of complex systems are difficult to predict (nonlinearities, discontinuities,…)
  - ➔ Hard to predict the future behavior (from past observations)
  - No universal prediction scheme, efficient with every signal
- Challenges: fast, causal and reliable prediction
  - Small computing cost
  - Accurate predictions for any signal behavior
- Idea: Borrow a context-based prediction,

commonly used in lossless image encoders,

(e.g. GIF or PNG)



Context-based extrapolation
Accuracy: relative integration error/com. step size

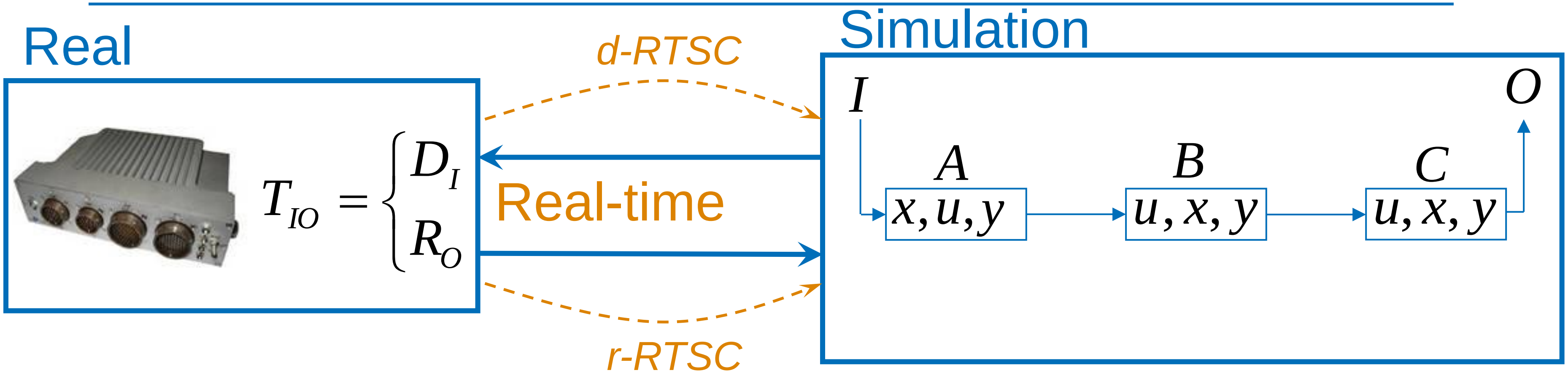


Outline

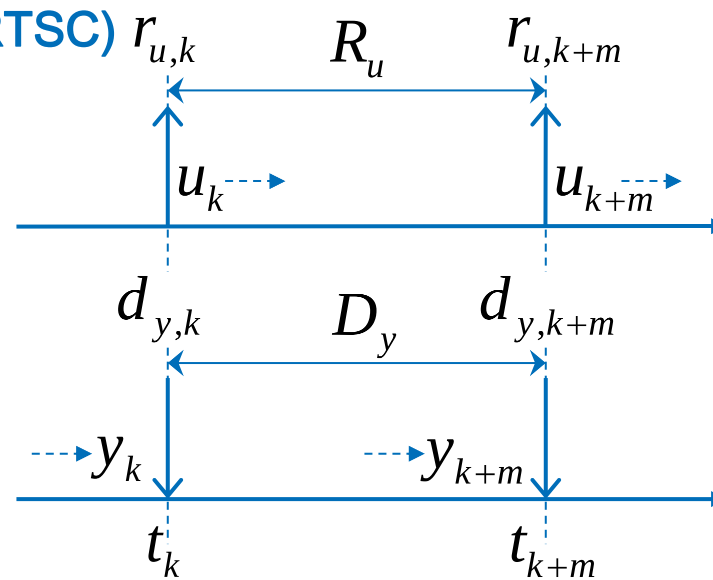
- Context
- Problem description
- IFPEN results on simulation acceleration
  - Splitting is speed-up
  - Ensuring speed-up and accuracy: the RCOSIM method
  - Context-based extrapolation
  - Mapping real-time constraints on a system-level simulation
- Future work

Real-time simulation

From real-time to simulated time



- **Relative Real-Time Simulation Constraints (RTSC)**
- **At each interaction :** $n \times T_{IO} = t_k$
  - y_k **required → *deadline*** $d_{y,k}$
  - u_k **available → *release*** $r_{u,k}$
- **Impact on the underlying computations**
  - **Related to the characteristics and interconnections**
  - **How to propagate?**



RTSC propagation

- Models graph
  - RTSC propagated to all (u, x, y)
  - Propagation rules (data flow)
    - Release : r-mesh, from start to end of the graph
    - Deadline : d-mesh, from end to start of the graph
  - ➔ $\forall t_k$: absolute constraints
- Confluent dependencies
- Heterogeneous dynamics
  - Time step $h_A \neq h_B$
    - Divisors of the period I/O
    - Multiples
    - Fixed
- Multiple I/O connections
- Cyclic graphs with restrictions

Intra-model propagation

	Non Direct Feedthrough	Direct Feedthrough
R	update_in $\xrightarrow{+h}$ update_out update_in $\xrightarrow{+h}$ update_state	update_in $\xrightarrow{0}$ update_out update_in $\xrightarrow{+h}$ update_state
D	update_out $\xrightarrow{-h}$ update_in update_out $\xrightarrow{0}$ update_state	update_out $\xrightarrow{0}$ update_in update_out $\xrightarrow{0}$ update_state

Phased RTSC constraints

$$R, D = (\Phi, T)$$

phase periode

Outline

- Context
- Problem description
- IFPEN results on simulation acceleration
  - Splitting is speed-up
  - Ensuring speed-up and accuracy: the RCOSIM method
  - Context-based extrapolation
  - Mapping real-time constraints on a system-level simulation
- Future work

Future work

- Address multi-rhythm models with RCosim
- Develop new dedicated heuristics
  - Handle non thread-safe implementation of FMU
  - Pipelining
- Define rules for fine-grained mapping of real-time constraints
  - Extend rules to handle RCosim level of granularity

Innovating for energy

www.ifpenergiesnouvelles.com