Fast multi-core co-simulation of Cyber-Physical Systems : Application to internal combustion engines

Abir Ben Khaled[a], Mongi Ben Gaid[a,*], Nicolas Pernet[a], Daniel Simon[b]

[a]*IFP Energies nouvelles, 1-4 avenue de Bois-Préau, 92852 Rueil-Malmaison, France*
[b]*INRIA/LIRMM (UMR 5506) - DEMAR team, 95 rue de la Galéra, 34090 Montpellier, France*

Abstract

The design process of complex Cyber-Physical Systems often relies on co-simulations of the system, involving the interaction of several simulated models of sub-systems. However, reaching real-time simulations is currently prevented by prohibitive CPU times using the single-threaded existing simulation tools. This paper investigates the problem of the efficient parallel co-simulation of hybrid dynamical systems. It introduces a finely-grained co-simulation method enabling numerical integration speed-ups. It is obtained using a partition across the model into loosely coupled sub-systems with sparse communication between modules. The proposed scheme leads to schedule a large number of operations with a wide range of execution times. A suitable off-line scheduling algorithm, based on the input/output dynamics of the models, is proposed to minimize the simulation errors induced by the parallel execution. This scheme is finally tested using the phenomenological model of a combustion engine issued from the Functional Mockup Interface framework. Compared with the sequential case, it shows significant speed-ups while keeping the numerical integration accuracy under control.

Keywords:
Hybrid dynamical system, Distributed simulation, Cyber-Physical System, Numerical integration, Model decomposition, Multi-core scheduling

*Corresponding author at: IFP Energies nouvelles, 1-4 avenue de Bois-Préau, 92852 Rueil-Malmaison, France. Tel.:+33147525029

Email addresses: abir.ben-khaled@ifpen.fr (Abir Ben Khaled), mongi.ben-gaid@ifpen.fr (Mongi Ben Gaid), nicolas.pernet@ifpen.fr (Nicolas Pernet), daniel.simon@inria.fr (Daniel Simon)

1. Introduction

A major challenge of the 21^{th} century is to succeed the energy transition, from an economy that is currently based on fossil energy, to an economy that relies on renewable energy and energy efficiency. This challenge affects the whole energy cycle: production, transport as well as consumption.

The transport sector consumes significant amounts of energy. It is predominantly reliant on oil, a resource that is limited and whose availability is expected to vanish during this century. Reducing fuel consumption and diversifying energy sources are major challenges in this field.

Automobiles are typical examples of Cyber-Physical Systems, where chemical energy (gasoline, diesel, ethanol fuel...) or electrical energy is converted to kinetic energy. Electronic controllers and networks present in vehicles interact with vehicles components that are sub-systems of multi-physical nature (mechanical, thermodynamic, electrical ...) and whose design involves multi-disciplinary teams.

In this design process, simulation is proven to be an indisputable step between concept design and prototype validation. Realistic simulations allow for the preliminary evaluation, tuning and possibly redesign of proposed solutions ahead of implementation, thus lowering the risks. To be confident in the result, building such simulations needs high fidelity models both for the components and for their interaction.

Currently, building high fidelity system-level models of Cyber-Physical Systems in general and automotive cars in particular, is a challenging duty. One problem is the diversity of modeling and simulation environments used by the various involved multi-disciplinary teams. Particular environments are preferred for a specific use due to distinctive strengths (modeling language, libraries, solvers, cost...). The Functional Mock-up Interface (FMI) specification has been proposed to improve this issue [1]. It is a tool independent and open standard[1] designed to support both the exchange and the co-simulation of dynamic models by combining software components provided by different sources. In particular, it was intended from the beginning to support the use of the AUTOSAR[2] standard and the Modelica language. The Model Exchange part of the standard provides the encapsulation of models equations in well defined components and interfaces. The Co-Simulation part allows for the coupling of several models together with

[1] `https://www.fmi-standard.org/`

[2] `http://www.autosar.org`

their solvers in a co-simulation environment, designed to manage the data exchange and synchronization between subsystems.

A second problem is the prohibitive CPU times observed when such high-fidelity models are run. This is due to the fact that major system-level simulation softwares are currently unable to exploit multi-core processors, because they are relying on sequential Ordinary Differential Equation (ODE) and Differential Algebraic equation (DAE) solvers. To solve this issue, several studies have shown that co-simulation approaches can provide significant improvements. Co-simulation allows to simulate together models coming from different areas and to validate both the individual behaviors and their interaction. The simulators may be exported from original authoring tools as Functional Mock-up Units (FMUs) and then imported in a co-simulation environment. Hence they cooperate at run-time thanks to the FMI definitions of their interfaces, and to the master algorithms of these environments. Moreover the "FMI for model exchange" framework [1] allows for solving independently the sub-models using custom solvers.

Both modeling and numerical integration deal with approximations, hence it is first needed to find a satisfactory trade-off between the simulation speed and precision. Ultimately, the simulation of the physical models will take into account some real-time constraints introduced by the interaction with the real components. Indeed, these models are intended to validate controllers, e.g., to combine high efficiency with clean combustion. Using hardware-in-the-loop (HIL) simulations means that the interaction between the simulated world and the real world must be consistent, i.e. that the simulated time and real-time must match at some precise points [2].

This paper proposes an approach allowing the speed-up and the parallel simulation of Cyber-Physical Systems modeled by hybrid ODEs and relying on the FMI specification, as well as a complex benchmark for its assessment. This approach can be seen as an intermediate method between simulation and co-simulation.

The paper is structured as follows. Section 2 surveys related works on several parallelization approaches. In Section 3, a partitioned model is formalized for co-simulation, then Section 4 describes the model of computation. In Section 5, we propose a co-simulation method that improves both the integration speed and the accuracy of the solution, compared to the existing method. Section 6 presents a case study using an internal combustion engine model, and several experiments are conducted in Section 7 to demonstrate the effectiveness of the proposed approach. Section 8 concludes this paper and discuss future work plans.

2. Related work

Burrage proposed a classification into three categories of the methods for the parallel solution of ODEs [3], that is still valid for hybrid ODEs or DAEs.

2.1. Parallelization across the method

The first idea is to exploit concurrent functions evaluations (like state derivatives) during the computation of each integration step.

Explicit multistage Runge Kutta methods are sequential. However, in some cases, two or more stages of Diagonally Implicit Runge Kutta methods can be executed in parallel, if some coefficients of the strictly lower triangular matrix associated to the method are zero. The parallelization of these methods was studied in [4]. The conclusion is that the parallelization potential is limited.

In the sixties, Miranker and Liniger [5] have proposed a framework that allows devising parallel predictor/corrector methods. Later generalizations are reviewed in the surveys by [6] and [7]. Like the previous approaches, these methods offer a limited parallelization potential. One drawback is their relatively small stability regions.

Other approaches that can fit this classification rely on parallelizing matrix inversions, which are needed when using an implicit method [8] or parallelizing operations on vectors for ODEs resolution by separating them into modules (see PVODE solver [9] implemented using MPI (Message-Passing Interface) technology).

2.2. Parallelization across the steps

In this approach, equations are solved in parallel over a large number of steps. It is based on a time decomposition method, originally introduced to solve Partial Differential Equations (PDEs) using the multi-grid approach [10]. Among the techniques introduced in this area, the Parareal scheme proposed in [11] and the PITA algorithm described in [12], where both of them were derived from the multiple shooting method [13]. They follow the approach of splitting the time domain in sub-domains by considering two levels of time grids. A first parallel computation of a predicted solution is performed with a fine time grid. After that, at each end of time of sub-domains, the solution makes a jump with the previous Initial Boundary Value (IBV) of the next time sub-domain. A correction of the IBV for the next fine grid is then computed on the coarse time grid. Nevertheless, this approach seems to have difficulties for stiff nonlinear problems.

2.3. Parallelization across the model

Numerically integrating PDEs in parallel is made easier by the need to solve across their spatial dimensions, which naturally leads to data parallelism. However, this is not the case for ODEs and DAEs, where both models and solvers exhibit a strong sequential nature. Decoupling this apparently sequential computation is an important issue for distributed simulation, because the way of decoupling a system could significantly affect the simulation results and speed. Some important methods tried to exploit the parallelization across the model.

Waveform relaxation [14] is a method that relies on Picard iteration approach for solving ODEs. It is based on a Jacobi or Gauss-Seidel iterative scheme in the framework of which each relaxation element is called a wave. The global system is partitioned into subsystems, which are solved separately and then, they exchange the computed waves. The algorithm is repeated until global convergence. The efficiency of this method, i.e. the convergence rate, depends on the quality of the system partitioning, and works better with weak coupling. In [15], three types of coupling methods were compared to show how they affect the convergence of the solution.

Transmission line modeling [16] is a discrete modeling method that provides a general approach for decoupling systems. It represents the physical process by a transmission-line graph. According to this method, the decoupling point should be chosen where variables change slowly. Consequently, the decoupled subsystems are seen as if they were connected by constant variables and the error due to time delays can be significantly decreased or even eliminated (when the communication step is chosen equal to the real physical delay of the decoupled components). The Hopsan [17] simulation tool, used primarily for hydro-mechanical simulation, natively implements this method.

Modular time-integration or co-simulation sees the system to be integrated as a connection of several subsystems. The data exchange between subsystems is restricted to the discrete synchronization points. Between these synchronization points the subsystems are integrated independently of each other. The numerical stability of these methods was studied in [18]. In [2], the application to the context of Hardware-In-The-Loop was studied, and several alternatives were proposed to perform real-time simulation of complex physical models. The study focused on the case of fixed time-step solvers, then it was extended in [19] to examine the case of variable time-step solvers. In [20], automatic decoupling techniques, based on incidence matrices, were studied to improve the parallel

performance of hybrid dynamical systems. Parallelization of Declarative Object-Oriented Models was studied in [21], and an approach for the decomposition into weakly coupled components was proposed in [22].

3. Motivation for multi-simulator approach

3.1. Model partition for numerical integration

Complex physical systems are generally modeled by hybrid non-linear ODEs or DAEs. The hybrid behavior is due to the discontinuities, raised by events triggered off when a given threshold is crossed (zero-crossing), and it plays a key role in the complexity and speed of the simulation. In fact, more the model has events and more the numerical integration is slowed down. This behavior is observed for both fixed and variable time-step solvers. Fixed time-step solvers cannot exactly catch the time of discontinuities, and the time-step must be chosen very small to come closer to the instant when an event occur. For variable time-step solver which do not have the ability for events detection, the integration time-step is decreased until reaching tiny values to capture the zero-crossing instant. For those with zero-crossing detection, the integration is anyway restarted anew at each event occurrence after an iterative event location procedure [23]. It appears that numerous discontinuities in the hybrid system sadly prevents variable step solvers to reach the high integration speeds which could be attained only considering the system's continuous dynamics [20].

However, it often appears that the incidence matrices between state variables, or between state variables and events, are sparse. Events are raised only by the evolution of a subset of the state vector, and the corresponding discontinuities only act upon a subset of the system. Thus, to improve the simulation speed, it is proposed to partition the model into sub-systems such that every discontinuity processing can be, as far as possible, encapsulated in a single sub-system.

Hence, each sub-system can be integrated by its own solver, avoiding interrupts coming from unrelated events. Moreover, events detection and location inside a sub-system involve a smaller variables set and can be processed faster.

Note that we are especially interested on the modular co-simulation approach. In fact, unlike the waveform relaxation technique, there are no iterations until convergence, which is more suitable for real-time and HIL simulation. In addition, compared to TLM, the communication step can be chosen different from the real physical delay of the decoupled components.

3.2. Model formalization

A formal model is now provided to support the co-simulation analysis. Consider an initial hybrid dynamical system Σ described by nonlinear differential equations :

$$\dot{X} = f(t, X, D, U) \quad \text{for } t_n \leq t < t_{n+1}, \tag{1a}$$

$$Y = g(t, X, D, U), \tag{1b}$$

where $X \in \mathbb{R}^{n_X}$ is the continuous state vector, $D \in \mathbb{R}^{n_D}$ is the discrete state vector, $U \in \mathbb{R}^{n_U}$ is the input vector, $Y \in \mathbb{R}^{n_Y}$ is the output vector and $t \in \mathbb{R}^+$ is the time.

$(t_n)_{n\geq 0}$ is a sequence of strictly increasing time instants representing discontinuity points called "state events", which are the roots of the equation

$$h(t, X, D, U) = 0. \tag{2}$$

h is usually called zero-crossing function or event indicator, used for *event detection* and *location* [23].

At each time instant t_n, a new continuous state vector may be computed as a result of the *event handling*

$$X(t_n) = I(t_n, X, D, U), \tag{3}$$

and a new discrete state vector may be computed as a result of discrete state update

$$D(t_n) = J(t_{n-1}, X, D, U). \tag{4}$$

If no discontinuity affects a component of $X(t_n)$, the right limit of this component will be equal to its value at t_n.

It is assumed that Σ is well posed in the sense that a unique solution exists for each admissible initial conditions $X(t_0)$ and $D(t_0)$ and that consequently X, D, U, and Y are piece-wise continuous functions in $[t_n, t_{n+1}]$.

To execute the system in parallel, the model must be split into several sub-models. For simplicity, assume that the system is decomposed into two separate blocks denoted model A and model B in Fig. 2. Our approach generalizes to any decomposition into N blocks of system Σ.

Therefore, the sub-systems can be written as:

$$\begin{cases} \dot{X}_A = f_A(t, X_A, D_A, U_A) \\ Y_A = g_A(t, X_A, D_A, U_A) \end{cases} \textit{ and } \begin{cases} \dot{X}_B = f_B(t, X_B, D_B, U_B) \\ Y_B = g_B(t, X_B, D_B, U_B) \end{cases} \tag{5}$$

with $X = [X_A \;\; X_B]^T$ and $D = [D_A \;\; D_B]^T$

Here U_A are the inputs needed for model A, directly provided by the outputs Y_B produced by model B. Similarly, U_B are the inputs needed for model B directly provided by the outputs Y_A produced by model A.

Note that when the simulators are run in parallel, using the co-simulation approach, they are periodically synchronized to exchange data at every communication time-step denoted T_c (Fig. 1). It is assumed that the synchronization steps are by far larger than the internal integration steps, so that each simulator integrates at its own rate and considers the data coming from other simulators are held as constant between the communication points.

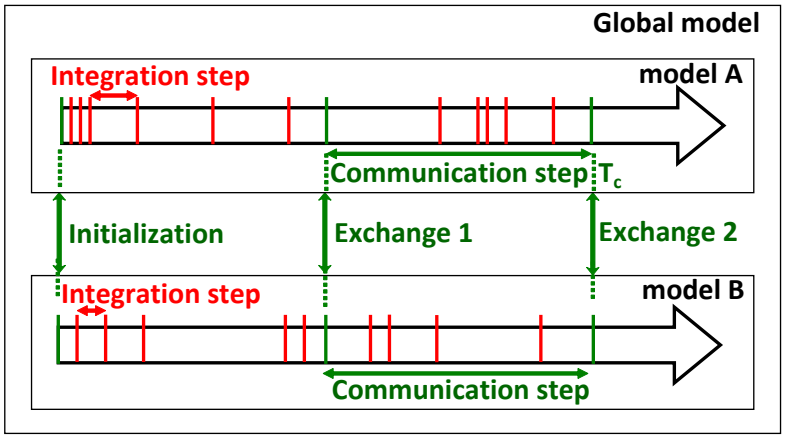

Figure 1: Periodic synchronization between simulators

N blocks resulting from system Σ are connected into an arbitrary diagram, each block M is assigned to a thread and a dedicated solver. In a preliminary approach, a common communication step-size T_c is shared by all blocks, so that they all read their inputs and update their outputs at communication points that are multiple of T_c.

3.3. Model decomposition and data dependencies cycles

To exploit the parallelism provided by the multi-core platform a physical model is partitioned into co-simulation components. However, the partitioning process may lead to the generation of dependency loops between the components (as in Fig. 2).

To start the co-simulation, these loops must be broken by choosing an execution order. Depending on this choice, it is possible that some outputs are delayed,

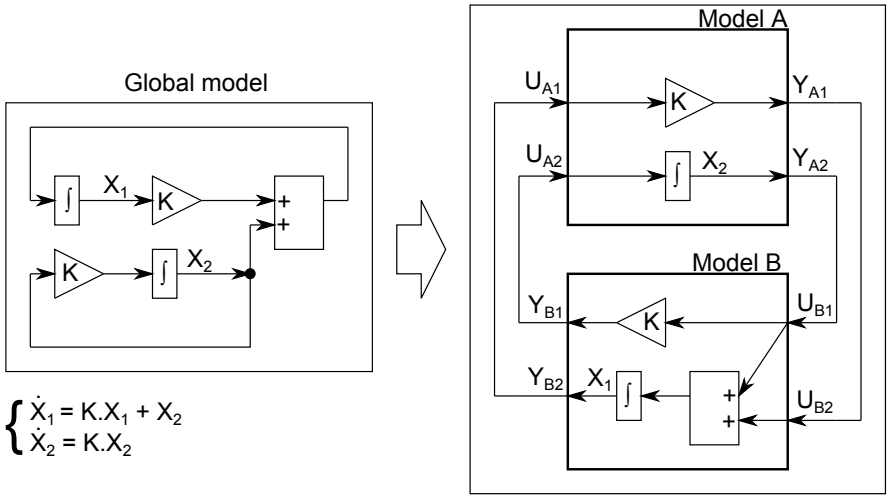

Figure 2: Loop creation due to the model splitting

thus inducing simulation errors. The next section examines how to break these loops.

4. Model of computation with the modular co-simulation approach

As mentioned in section 3.3, breaking the created loop may lead to delayed outputs, depending on the models input/output properties. A model is Non Direct Feedthrough (NDF) when all its outputs depend only on its state vector : $Y = g(t, X)$. It is Direct Feedthrough (DF) if at least one of its outputs is a direct function of the inputs : $Y = g(t, X, U)$. Two cases are considered.

4.1. 1st case: Coexistence of DF and NDF models

In this case, model A and model B are respectively considered NDF and DF. Since the initial conditions ($k = 0$) of $X_A(t_k)$ and $X_B(t_k)$ are known, only the outputs $Y_A(t_k)$ (and consequently $U_B(t_k)$) are ready to be computed. After their calculation, $\dot{X}_B(t_k)$ and $Y_B(t_k)$ (and consequently $U_A(t_k)$) are ready to run. Once $U_A(t_k)$ is available, the computing of $\dot{X}_A(t_k)$ is ready to be run. The same cycle is repeated until the end of simulation (Fig. 3).

In fact the loop is not algebraic because the execution order is naturally defined. Therefore, the delay is avoided when starting with the NDF models, i.e. using NDF→DF order. In other words the whole (update outputs/update states)

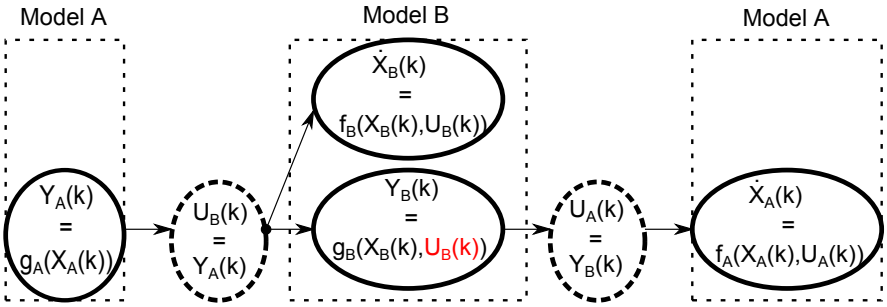

Figure 3: Defined execution order between NDF and DF models

procedure takes place in a single instant ($t = t_k$) of the simulated time. Obviously, it is enough to have one NDF model in the loop to prevent the outputs from being delayed by a causality cycle.

4.2. 2nd case: All the models are DF

In this case, model A and model B are both DF. At initial conditions ($k = 0$), $X_A(t_k)$ and $X_B(t_k)$ are known but none of $Y_A(t_k)$, $Y_B(t_k)$, $\dot{X}_A(t_k)$ and $\dot{X}_B(t_k)$ are ready to be computed. Indeed, $Y_A(t_k)$ and $\dot{X}_A(t_k)$ need $U_A(t_k) = Y_B(t_k)$, and at the same instant $Y_B(t_k)$ and $\dot{X}_B(t_k)$ need $U_B(t_k) = Y_A(t_k)$. This is a deadlock configuration and the loop is called algebraic. An execution order between A and B must be specified.

Regardless the execution order, it is inevitable to have at least a delayed model corresponding to the first executed one (Fig. 4). In fact, breaking the algebraic loop means that the link between the two models is replaced by a delay equal to the communication time-step T_c.

However, by having a good knowledge on the models, the delay-induced errors may be reduced. In fact, knowing if the outputs of a model are weakly or strongly coupled to its inputs and/or if they are slowly (e.g. pressure, temperature) or rapidly changing may help to determine an efficient execution order. It is interesting in that case to begin by the model where the majority of its outputs are weakly coupled to its inputs and/or that are changing smoothly because its behavior can be assimilated to a NDF or a weak DF model. Nevertheless, even if the delay-induced errors are reduced, they cannot be totally eliminated.

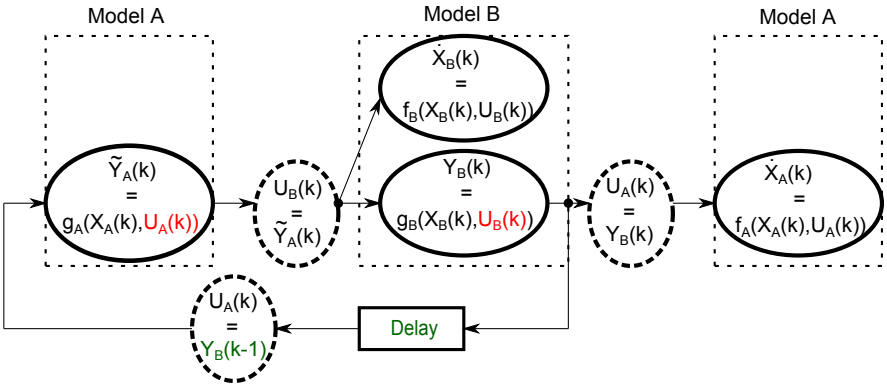

Figure 4: Delayed outputs due to the breaking of the algebraic loop

5. Proposed model of computation: The refined co-simulation approach

5.1. Refined dependency graph with the FMI/FMU specification

Even with an efficient execution order, when all the models are DF, delayed outputs still exist in the modular co-simulation approach. To take advantage of the model splitting without adding useless delays, it deserves to look at the problem with a refined viewpoint. Thanks to the FMI specifications, it is possible to access information about the relationships between inputs and outputs inside a model encapsulated in a FMU (Fig. 5).

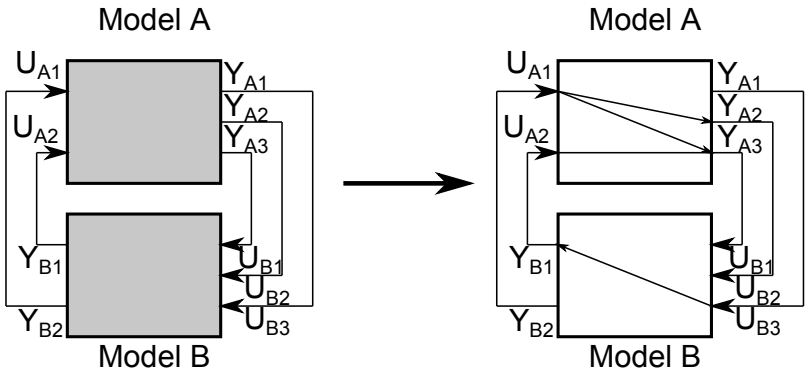

Figure 5: Input/Output connection through intra and inter models view

Therefore the co-simulation processing can be refined. Instead of considering the entire module as DF or NDF, it is possible with FMI to sort the outputs by

identifying locally if they are DF or NDF. For example, in Fig. 5, model A and model B are both DF at the module level. Exploring the input and output links inside each model reveals there is no cycle which contains only DF outputs. Furthermore, a FMU provides different functions to compute each output separately (i.e. components of (1b)), and a specific one to update the model states (i.e. integrate (1a)). By knowing both intra and inter model dependencies between inputs and outputs, these functions allow various execution possibilities without a strict model execution order. The parallelization granularity is increased and the distribution of the different operations among the different processors becomes a more complex problem.

A co-simulation of the different FMUs with constant communication steps can be described by a directed graph where vertices are operations and edges are precedence relations between these operations. Moreover, knowing that the global model is described by ODEs and does not present algebraic loop, such graph is necessarily a Direct Acyclic Graph (DAG) (Fig. 6). More precisely, operations are either update output ($update_{\text{out}}$), update input ($update_{\text{in}}$) or update state ($update_{\text{state}}$) function calls. An edge from an $update_{\text{out}}$ to an $update_{\text{in}}$ corresponds to an inter model data dependency (for example from Y_{B2} to U_{A1} in Fig. 6). These edges are the expression of the data dependencies between the models. An edge from an $update_{\text{in}}$ to an $update_{\text{out}}$ expresses an intra model DF dependency (for example from U_{B3} to Y_{B1} in Fig. 6). These dependencies are listed in each model FMU. There is an edge from each $update_{\text{in}}$ to the $update_{\text{state}}$ of the same model (for example from U_{A2} to $\dot{X}_{\text{A}}$ in Fig. 6), which means that all model inputs are necessary to update the state of the model. Finally, there is an edge from each $update_{\text{out}}$ to the $update_{\text{state}}$ of the same model (for example from Y_{A3} to $\dot{X}_{\text{A}}$ in Fig. 6), because the computation of $Y(t_k)$ needs $X(t_k)$ which is no longer available after $update_{\text{state}}$ computed $X(t_{k+1})$. To run a co-simulation, each co-simulation step needs the whole DAG execution. Nevertheless the previous DAG execution must be totally finished before beginning the new one.

5.2. Scheduling heuristic

To achieve fast multi-core simulation, operations must be distributed and scheduled among the different available cores. To be effective, the distributed schedule must take into account the time cost of each operation. We propose to use an offline heuristic approach similar to the one of [24]. The heuristic considers start and end dates for each operation and tends to minimize the critical path latency of a DAG, in which a computation time is attached to each operation. For real-time simulation purpose, these computation times are estimated by their Worst Case

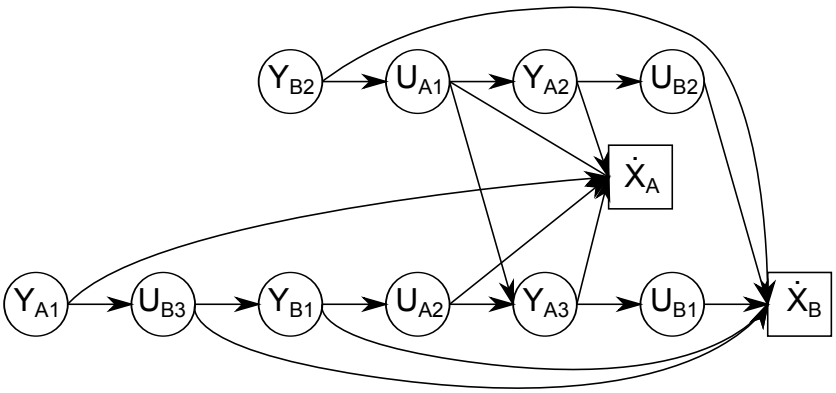

Figure 6: Refined dependency graph

Execution Time. Here, the goal is fast simulation, which is not safety critical, so that an average computation time (e.g. of a single-core simulation benchmark) can be used. In practical examples, the $update_{\text{state}}$ operations are by far more costly than $update_{\text{out}}$, while $update_{\text{in}}$ are simple data copy whose cost is negligible.

The heuristic cost function computes the schedule pressure of a given operation on a specific core. This schedule pressure is the difference between the critical path increase (by setting this operation on this core) and the operation flexibility (difference between its earliest start time and its latest end time). At each step, for each remaining operation for which all predecessors have already been scheduled, the heuristic computes the schedule pressure of this operation on each core, and sets this operation on its best core, i.e. the one which minimizes the pressure. Then, among all the pending operations, the one with the largest pressure (on its best core) is selected and added to the schedule.

Nevertheless, there are other operation allocation constraints. Indeed, FMI standard does not force a FMU operation to be thread safe and currently, the FMU operations $update_{\text{out}}$ calls cannot be performed in parallel. Because this constraint might be relaxed either with a next FMI version or from another FMU tool, it is decided to temporarily reduce the heuristic search space. All the operations related to a given FMU are bind to the same core, which is the one elected by the heuristic for the first scheduled operation of the given FMU. Each time an operation is scheduled, synchronization operations are inserted if needed. For example if Y_{A1} and U_{B3} are allocated by the heuristic on different cores, a semaphore is signalled just after Y_{A1} on its core, and a waiting semaphore operation is executed on the other core just before U_{B3}.

Using the computation time C_i of each operation OP_i, the first step for the scheduling heuristic is to compute the start and end dates from the graph start denoted S_i and E_i, then the critical path $\mathrm{CP} := \max_i E_i$. After that, the start and end dates from the graph end denoted S_i^* and E_i^* and then the flexibility $F_i := \mathrm{CP} - E_i - E_i^*$ can be performed. Finally, the heuristic incrementally builds the scheduling by defining the best core allocation for each ready operation and then by selecting the one with the maximal cost (see Algorithm 1).

Compared to distributed co-simulation approaches with a model-based granularity, the refined approach has two important advantages. First, using a finer granularity potentially increases the models decoupling possibilities and allows to reach increased co-simulation speed-up. Second, dependencies between the models inputs and outputs are satisfied through both inter and intra model dependencies, allowing to find a valid schedule without inserting useless delays. It makes the co-simulation results closer to the reference simulation ones. The next section illustrates these advantages on an powertrain case study.

6. Case study

6.1. Engine simulator

In this study, a Spark Ignition (SI) RENAULT F4RT engine has been modeled. It is a four-cylinder in line Port Fuel Injector (PFI) engine in which the engine displacement is $2000\,\mathrm{cm}^3$. The air path is made of a turbocharger with a mono-scroll turbine controlled by a waste-gate, an intake throttle and a downstream-compressor heat exchanger.

The engine model was developed using ModEngine library [25]. ModEngine is a Modelica [26] library that allows for the modeling of a complete engine with diesel and gasoline combustion models. The engine model and the split parts were imported into xMOD model integration and virtual experimentation tool [27], using the FMI export features of Dymola.

The engine model has 118 state variables and 312 event indicators (of discontinuities).

6.2. Decomposition approach

The partitioning of the engine model is performed by separating the four-cylinder from the air path (AP), then by isolating the cylinders ($C_i/i = 1..4$) from each other. This kind of splitting allows for the reduction of the number of events acting on each sub-system. In fact, the combustion phase raises most of the events,

```
Initialization;
Set Ω the set of all the operations;
Set Γ the set of all the available cores;
foreach OP_i ∈ Ω do
    Set FixedCore_i := NOT_ALLOCATED; (operation OP_i is not already allocated);
end
foreach Core_j ∈ Γ do
    Set T_Core_j := 0; (where T_Core_j corresponds to the first idle time on Core_j);
end
Set O the set of operations without predecessors;
while O ≠ ∅ do
    foreach OP_i ∈ O do
        if FixedCore_i == NOT_ALLOCATED then
            Set cost_i to ∞; (cost of OP_i is set to the maximum value);
            foreach Core_j ∈ Γ do
                S'_i := max(S_i, T_Core_j); (new start date of OP_i when executed on Core_j);
                cost_i,j := S'_i + C_i + E*_i − CP; (cost of OP_i when executed on Core_j);
                if cost_i,j < cost_i then
                    Set cost_i := cost_i,j;
                    Set BestCore_i := Core_j;
                end
            end
        else
            Set BestCore_i := FixedCore_i;
            S'_i := max(S_i, T_Core_BestCore_i);
            cost_i := S'_i + C_i + E*_i − CP;
        end
    end
    Find OP_i with maximal cost_i in O;
    Schedule OP_i on its core BestCore_i;
    Set k := BestCore_i;
    T_Core_k := T_Core_k + C_i; (Advance the time of Core_k);
    if OP_i is the first operation scheduled for its FMU then
        foreach OP_j of this FMU do
            FixedCore_j := BestCore_i;
        end
    end
    Remove OP_i from the set O;
    Add to the set O all successors of OP_i for which all predecessors are already
    scheduled;
end
```

Algorithm 1: Scheduling heuristic: minimization of cost function.

which are located in the firing cylinder. The solver can process them locally during the combustion cycle of the isolated cylinder, and then enlarge its integration time-step until the next cycle.

From a thermodynamic point of view, the cylinders are loosely coupled, but a mutual data exchange does still exist between them and the air path. Since the dynamics of the air path is slow (it produces slow outputs to the cylinders, e.g. temperature) and those of the cylinders are fast (they produce fast outputs to the air path, e.g. torque), the execution order of the split model is chosen from air path to cylinders in accordance with the analysis in section 4.2

The model is divided into 5 components and governed by a basic controller denoted CTRL. It gathers 91 inputs and 98 outputs. The scheduling of the refined co-simulation approach deals with 103 operations (5 $update_{\text{state}}$ and 98 $update_{\text{out}}$).

6.3. Models of computation

This study compares the simulation performance, observing the trade-offs between the simulation speed and simulation accuracy, for the following approaches:

- Simulation of the whole engine model in a single thread using a single solver, to provide the reference for precision evaluations.

- Modular co-simulation of the split model with respect to data dependencies. This is the standard version of the modular co-simulation, denoted "sv-MCosim", where the execution order is fixed from models with slow changing outputs to those with fast changing outputs. For the case study, all the cylinders must wait for the execution of the AP.

- Modular co-simulation of the split model with broken data dependencies. This is the extended version of the modular co-simulation approach, denoted "ev-MCosim", where all the data dependencies are relaxed. For the case study, the AP and all the cylinders are integrated in parallel during each communication interval.

- Refined co-simulation of the split model, this method is denoted "RCosim". For the case study, all the inputs and the outputs are updated following the order of scheduling heuristic, then the integration of the AP and the cylinders are performed in parallel.

These methods are sketched in Fig. 7, where DT is the execution time during a T_c. DT gathers the integration of the models in the blocks X_i (e.g. X_{AP} for AP) and

the input and output updates, as in the IN/OUT blocks for modular co-simulation and in the IN/OUT/WAIT blocks for refined co-simulation (the waiting times are introduced by the scheduling heuristic). The models are simulated on separate cores using their own solver.

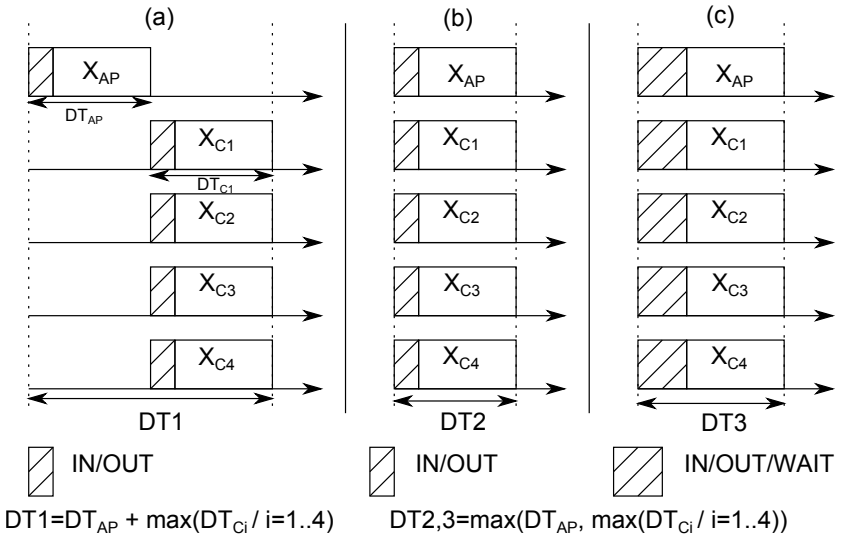

Figure 7: (a) sv-MCosim method; (b) ev-MCosim method; (c) RCosim method

7. Tests and results

Tests are performed on a platform with 16 GB RAM and 2 "Intel Xeon" processors, each running 8 cores at 3.1 GHz.

7.1. Model validation

The model validation is based on the observation of the intake and exhaust manifold pressures (Pman, Pexh), air-fuel equivalence ratio (AFR) and torque. These outputs are computed using LSODAR which is a variable time-step solver with a root-finding capability to detect the events occurring during the simulation. It has also the ability to adapt the integration method depending on the observed system stiffness.

The simulation reference Y_{ref} is built from the integration of the entire engine model, the solver tolerance (tol) being decreased until reaching stable results, which is given for tol = 10^{-7} at the cost of a slow simulation speed.

Then, to explore usable trade-offs between the simulation speed and precision, the relative integration error Er (defined in (6)) is set to be less than 1 %.

$$Er(\%) = \frac{100}{N} \cdot \sum_{i=0}^{N-1} \left(\left| \frac{Y_{\text{ref}}(i) - Y(i)}{Y_{\text{ref}}(i)} \right| \right) \tag{6}$$

with N the number of saved points during 1s of simulation.

Iterative tests show that the desired error ($Er \leq 1\,\%$) can be reached with tol = 10^{-4} (Table. 1). Using the split model, each of its 5 components is assigned to a dedicated core and integrated by LSODAR with tolerance tol = 10^{-4}.

Table 1: Relative integration error

Outputs	Pman	Pexh	Torque	AFR
Er(%)	0.027	0.05	0.38	0.37

The modular co-simulation executes for each model all the $update_{\text{out}}$ operations in one single block as for the $update_{\text{state}}$ operation. Fig. 8 illustrates the time-chart and shows the waiting period on the AP which it represents the difference between sv-MCosim and ev-MCosim.

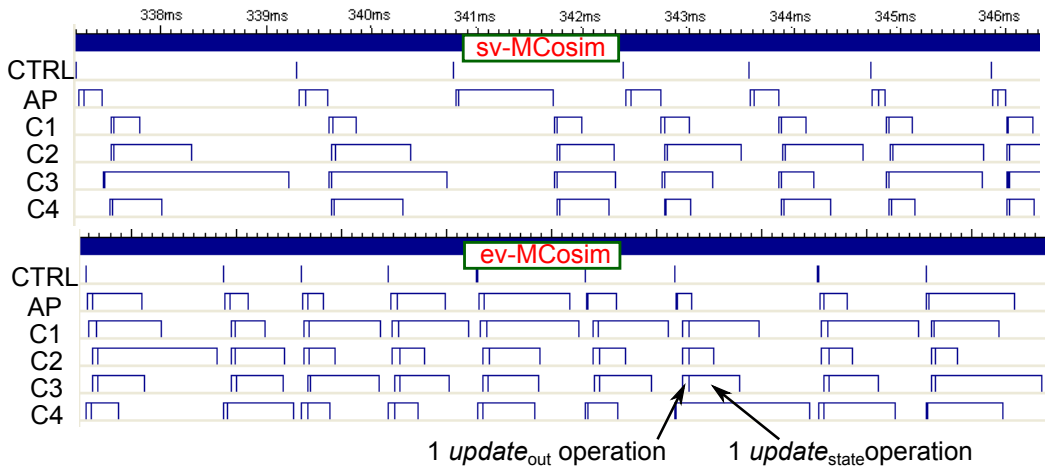

Figure 8: Modular co-simulation time-chart

The refined co-simulation schedules the 103 $update_{\text{out}}$ and $update_{\text{state}}$ operations. As described in Fig. 9, the computation time of $update_{\text{out}}$ are negligible

compared to $update_{\text{state}}$. A zoom on the time-chart shows the scheduling of the $update_{\text{out}}$ operations.

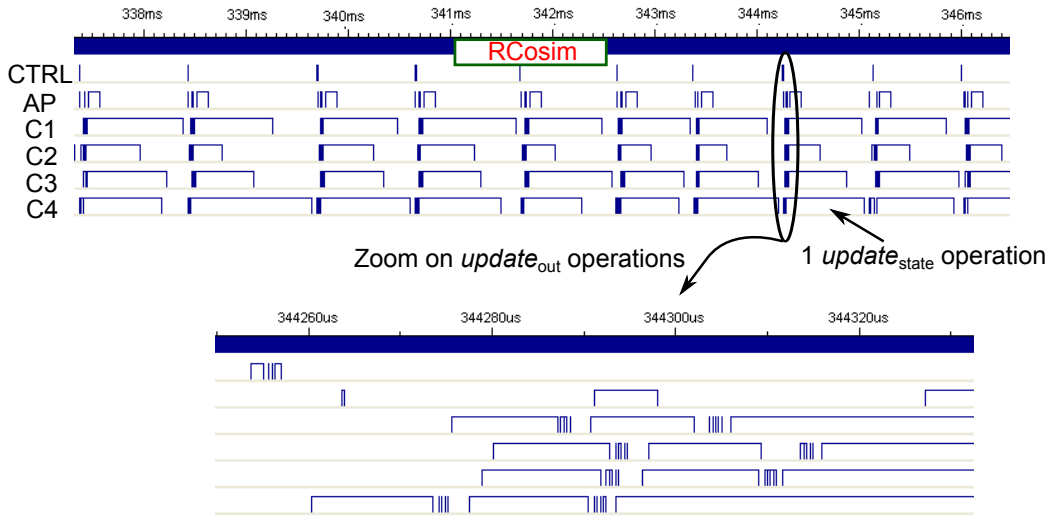

Figure 9: Scheduling of the update operations with RCosim

7.2. Accuracy tests

7.2.1. DF outputs

The torque is a DF output of the air path. Test results show that the torque is delayed by T_c with the modular co-simulation method, as expected since all the models are DF. However, thanks to the refined co-simulation method, the torque is almost identical to the reference as indicated in Fig. 10.

Then, the relative integration error is computed for several communication steps as in Table. 2. The results show that the refined co-simulation method keeps the integration stable even for large T_c. In fact, Er stays close to 1 %, whereas the modular co-simulation method suffers from delay-induced errors up to almost 20 %.

7.2.2. NDF outputs

The manifold pressure is a NDF output of the air path. For this case, there is no delay whatever the method. As for the torque, the relative integration error of the pressure also depends on T_c (Fig. 11). However, the step width is not so harmful as there are not loop-induced delays.

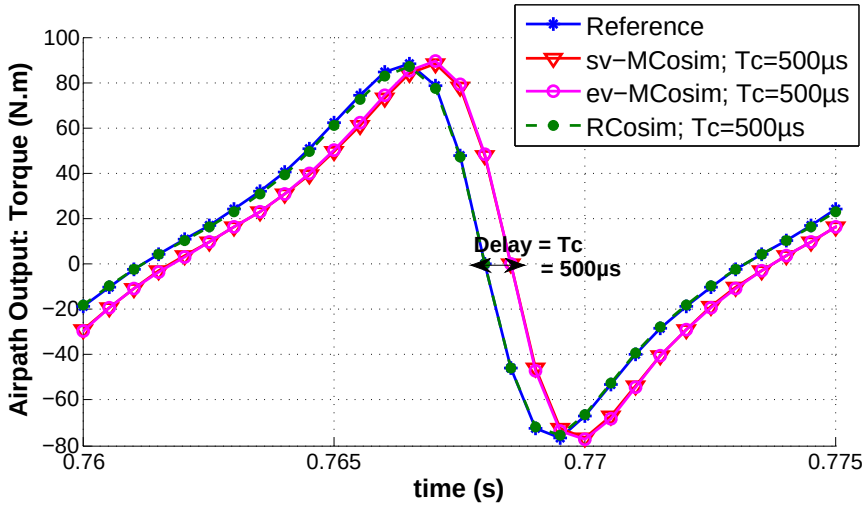

Figure 10: The behavior of a DF output regarding the different methods

Table 2: Relative integration error on the (DF) torque

Simulation method	sv-MCosim	ev-MCosim	RCosim
Er(%) with $T_c = 100\,\mu s$	2.95	4.38	0.68
Er(%) with $T_c = 250\,\mu s$	9.12	9.33	1.1
Er(%) with $T_c = 500\,\mu s$	19.83	19.19	1.37

Table. 3 shows that the refined method is again advantageous for the simulation accuracy. To approach the desired error both for DF and NDF outputs, T_c must be restricted to $100\,\mu s$ for modular co-simulation whereas it can be enlarged up to $500\,\mu s$ with refined co-simulation.

Table 3: Relative integration error on the (NDF) manifold pressure

Simulation method	sv-MCosim	ev-MCosim	RCosim
Er(%) with $T_c = 100\,\mu s$	0.61	0.63	0.5
Er(%) with $T_c = 250\,\mu s$	1.2	1.11	0.88
Er(%) with $T_c = 500\,\mu s$	1.8	1.75	1.23

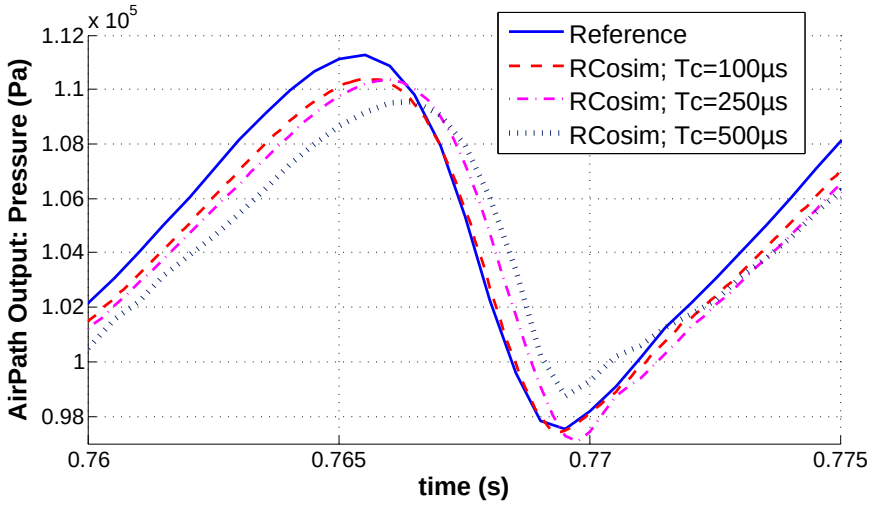

Figure 11: Effect of T_c on a NDF output for RCosim method

7.3. Simulation time speed-up

The integration of the engine model (118 state variables and 312 event indicators) is time-consuming. With tol $= 10^{-4}$, the sequential simulation on a single-core is 76.5 times slower than real-time. Compared with the reference case, speed-ups have been measured for $T_c = 250\,\mu s$ (to keep $Er \approx 1\,\%$). Table. 4 show that the speed-up reaches 7.82 for *sv-MCosim* and 8.84 for *ev-MCosim* (with relaxed dependencies between the air path and cylinders). The largest speed-up is gained with the refined co-simulation method and reaches 10.87, so that the simulation speed is now only 7.04 times slower than real-time. In fact, while integrating with the right (undelayed) input values at each T_c, the variable time-step solver rapidly finds the largest possible integration step. Indeed this later speed-up cannot be observed using fixed-step solvers.

Table 4: Simulation speed-up with the different approaches

Simulation method	sv-MCosim	ev-MCosim	RCosim
Speed-up	7.82	8.84	10.87

It is remarkable that, in all cases, the usual execution time penalty due to the multi-threading and distribution on 5 cores is greatly overcompensated by the

gains due to the wise partition across the original model.

7.4. Critical analysis of the results

The refined co-simulation technique showed significant improvements on results accuracy without deteriorating the speed-up provided by the modular co-simulation. This new technique allows a non-expert user to split a complex dynamic model without care of the execution order of the different blocks nor how to cut. In fact, by preserving a fine-grained execution order between $update_{\text{out}}$ operations, the integration and the states computation can be performed in parallel without implying delays on DF outputs. However, the efficiency of this approach lies in the assumption that the states integration is by far time consuming compared to the outputs computation. This assumption was based on work experience and major encountered industrial models and it was verified for the engine model.

To generalize the test results to any hybrid dynamical model, it is necessary to take into consideration the limits of the method's efficiency. This is reached when the sum of all the computation times of $update_{\text{out}}$ operations –that belong to the critical path CP– is approaching the computation time of an $update_{\text{state}}$ operation. In that case, the cost of $update_{\text{out}}$ operations cannot longer be neglected. However, it can always be compensated by the use of variable-time solver and the speed-up will be then similar to *ev-MCosim*.

Finally, our refined co-simulation proposed method aims at improving the accuracy of the existent modular co-simulation, assuming that a suitable split system is already provided. In fact, an associated system decomposition methodology is given in a previous work ([19]) that explain why and how the relaxation of the number of discontinuities (by splitting the hybrid dynamical system) improves the computation time and provides a supra-linear speedup. Then in [20], a system splitting using the block-diagonal form of incidence matrices, related to states and events was studied and applied to a mono-cylinder model which is not intuitive to split.

8. Summary and Perspectives

Cyber-Physical Systems gather a large number of physical and digital components, and simulation appears to be a key tool for their design and early validation. Beyond their size, their simulation is made difficult by their hybrid behavior combining ODE/DAEs with events, which prevents the usual sequential single-threaded simulation tools to reach fast simulation speeds.

Co-simulation allows to simulate together models coming from multi-disciplinary areas. This modular decomposition, based on the structure of the engineering system, provides a natural way to parallelize the original system across the model. More precisely, an efficient decomposition is intended to decouple the sub-models and minimize, as far as possible, their data and events related dependencies. Thus each model can be integrated by its own solver using an optimal integration step size, while data exchange is only possible at precise communication points. A suitable off-line scheduling of operations, taking care of the models I/O dynamics, allows for supra-linear speed-ups of simulations on a multi-core architecture, while keeping the simulation precision under control.

In practical applications, current co-simulation set-ups use a constant communication grid shared by all the models. Further improvements are expected from adaptive communication step sizes which may better handle the various changing dynamics of the models [28]. The size of the communication steps has a direct impact on the simulation errors, and effective communication step control should rely on on-line estimations of the errors induced by slackened exchange rates. Data extrapolation over steps is also expected to enhance the simulation precision over large communication steps. Indeed the stability of multi-rate simulators with adaptive steps needs to be carefully assessed, for example based on recent works on errors propagation inside modular co-simulations [18].

Finally, this study focused on numerical solvers based on time discretization and it is expected for the future to compare their efficiency to those based on state quantization since they are suitable for discontinuous ODEs [29].

References

[1] T. Blochwitz, T. Neidhold, M. Otter, M. Arnold, C. Bausch, M. Monteiro, C. Clauß, S. Wolf, H. Elmqvist, H. Olsson, A. Junghanns, J. Mauss, D. Neumerkel, J.-V. Peetz, The functional mockup interface for tool independent exchange of simulation models, in: 8th Int. Modelica Conf., Linköping Univ. Electronic Press, Dresden, Germany, 2011. doi:10.3384/ecp11063105.

[2] C. Faure, M. Ben Gaïd, N. Pernet, M. Fremovici, G. Font, G. Corde, Methods for real-time simulation of cyber-physical systems : Application to automotive domain, in: 7th Int. Wireless Communications and Mobile Computing Conf. IWCMC, Istanbul, Turkey, 2011, pp. 1105 – 1110. doi:10.1109/IWCMC.2011.5982695.

[3] K. Burrage, Parallel and Sequential Methods for Ordinary Differential Equations, Oxford Science Publications, 1995.

[4] A. Iserles, S. Nørsett, On the theory of parallel Runge-Kutta methods, IMA J. Numer. Anal. 10 (1990) 463–488.

[5] W. L. Miranker, W. Liniger, Parallel methods for the numerical integration of ordinary differential equations, J. Math. Comput. 21 (1967) 303–320.

[6] K. R. Jackson, A survey of parallel numerical methods for initial value problems for ordinary differential equations, IEEE Trans. Magn. 27 (1991) 3792–3797.

[7] K. Burrage, Parallel methods for initial value problems, Appl. Numer. Math. 11 (1993) 5–25.

[8] P. J. van der Houwen, B. P. Sommeijer, Parallel iteration of high-order Runge-Kutta methods with stepsize control, J. Comput. Appl. Math. 29 (1990) 111–127.

[9] G. D. Byrne, A. C. Hindmarsh, PVODE, an ODE solver for parallel computers, Int. J. High Perform. Comput. Appl. 13 (1999) 254–365.

[10] G. Horton, S. Vandewalle, A Space-Time Multigrid Method For Parabolic PDEs, Technical Report, Univ. Erlangen, 1993.

[11] J. Lions, Y. Maday, G. Turinici, Résolution d'EDP par un schéma en temps "pararéel", Comptes Rendus de l'Académie des Sciences - Series I - Mathematics 332 (2001) 661–668.

[12] C. Farhat, M. Chandesris, Time-decomposed parallel time-integrators : Theory and feasibility studies for fluid, structure, and fluid-structure applications, Int. J. Numer. Meth. Eng. 58 (2003) 1397–1434.

[13] P. Deuflhard, Newton Methods for Nonlinear Problems, volume 35 of *Springer Series in Computational Mathematics*, Springer, 2004.

[14] E. Lelarasmee, A. E. Ruehli, A. L. Sangiovanni-Vincentelli, The waveform relaxation method for time-domain analysis of large scale integrated circuits, IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. 1 (1982) 131–145.

[15] S. Y. R. Hui, C. Christopoulos, Numerical simulation of power circuits using transmission-line modelling, IEE proceedings. Part A. Physical science, Measurements and instrumentation, Management and education, Reviews 137 (1990) 379–384.

[16] M. Sjölund, R. Braun, P. Fritzson, P. Krus, Towards efficient distributed simulation in Modelica using Transmission Line Modeling, in: 3rd Workshop on Equation-Based Object-Oriented Modeling Languages and Tools EOOLT, Linköping Univ. Electronic Press, 2010, pp. 71–80.

[17] B. Eriksson, P. Nordin, P. Krus, Hopsan NG, a C++ implementation using the TLM simulation technique, in: 51th Conf. on Simulation and Modelling SIMS, Oulu, Finland, 2010.

[18] M. Arnold, Stability of sequential modular time integration methods for coupled multibody system models, J. Comput. Nonlin. Dyn. 5 (2010).

[19] A. Ben Khaled, M. Ben Gaïd, D. Simon, G. Font, Multicore simulation of powertrains using weakly synchronized model partitioning, in: IFAC Workshop on Engine and Powertrain Control Simulation and Modeling ECOSM, Rueil-Malmaison, France, 2012, pp. 448–455. doi:10.3182/20121023-3-FR-4025.00018.

[20] A. Ben Khaled, M. Ben Gaïd, D. Simon, Parallelization approaches for the time-efficient simulation of hybrid dynamical systems : Application to combustion modeling, in: 5th Workshop on Equation-Based Object-Oriented Modeling Languages and Tools EOOLT, Linköping Univ. Electronic Press, Nottingham, UK, 2013, pp. 27–36.

[21] F. Casella, A strategy for parallel simulation of declarative object-oriented models of generalized physical networks, in: 5th Workshop on Equation-Based Object-Oriented Modeling Languages and Tools EOOLT, Linköping Univ. Electronic Press, Nottingham, UK, 2013, pp. 45–51.

[22] A. V. Papadopoulos, A. Leva, Automating dynamic decoupling in object-oriented modelling and simulation tools, in: 5th Workshop on Equation-Based Object-Oriented Modeling Languages and Tools EOOLT, Linköping Univ. Electronic Press, Nottingham, UK, 2013, pp. 37–44.

[23] F. Zhang, M. Yeddanapudi, P. Mosterman, Zero-crossing location and detection algorithms for hybrid system simulation, in: 17th IFAC

World Congress, Seoul, South Korea, 2008, pp. 7967–7972. doi:10.3182/20080706-5-KR-1001.01346.

[24] T. Grandpierre, Y. Sorel, From algorithm and architecture specification to automatic generation of distributed real-time executives : A seamless flow of graphs transformations, in: 1st ACM/IEEE Int. Conf. on Formal Methods and Models for Codesign MEMOCODE, Mont Saint-Michel, France, 2003, pp. 123–132. doi:10.1109/MEMCOD.2003.1210097.

[25] Z. Benjelloun-Touimi, M. Ben Gaïd, J. Bohbot, A. Dutoya, H. Hadj-Amor, P. Moulin, H. Saafi, N. Pernet, From physical modeling to real-time simulation : Feedback on the use of modelica in the engine control development toolchain, in: 8th Int. Modelica Conf., Dresden, Germany, 2011.

[26] P. Fritzson, Principles of Object-Oriented Modeling and Simulation with Modelica, Wiley-IEEE Computer Society Pr, 2003.

[27] M. Ben Gaïd, G. Corde, A. Chasse, B. Léty, R. De La Rubia, M. Ould Abdellahi, Heterogeneous model integration and virtual experimentation using xMOD : Application to hybrid powertrain design and validation, in: 7th EUROSIM Congress on Modeling and Simulation, Prague, Czech Republic, 2010.

[28] T. Schierz, M. Arnold, C. Clauß, Co-simulation with communication step size control in an FMI compatible master algorithm, in: 9th Int. Modelica Conf., Munich, Germany, 2012, pp. 205–214. doi:10.3384/ecp12076205.

[29] G. Migoni, M. Bortolotto, E. Kofman, F. E. Cellier, Linearly implicit quantization-based integration methods for stiff ordinary differential equations, Simulation Modelling Practice and Theory 35 (2013) 118 – 136.