Distributed real-time simulation of numerical models : application to power-train

Abir Ben Khaled, Abir El Feki Ben Khaled-El Feki

UNIVERSITÉ DE GRENOBLE

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Automatique Productique**

Arrêté ministériel : 7 août 2006

Présentée par

Abir BEN KHALED - EL FEKI

Thèse dirigée par **Daniel SIMON**
et codirigée par **Mongi BEN GAID**

préparée au sein de **IFP Energies nouvelles et INRIA**
dans **l'École Doctorale EEATS**

Simulation temps-réel distribuée de modèles numériques: application au groupe motopropulseur

Distributed real-time simulation of numerical models: application to powertrain

Thèse soutenue publiquement le **27 mai 2014**,
devant le jury composé de :

Olivier SENAME
Professeur, Grenoble INP, Président
Petter KRUS
Professeur, Linköping University, Rapporteur
Pierre SIRON
Professeur, ISAE Toulouse, Rapporteur
Thierry SORIANO
Professeur, Supmeca Toulon, Examinateur
Ramine NIKOUKHAH
Directeur développement logiciel, Altair Development France, Examinateur
Daniel SIMON
Chargé de recherche, INRIA Sophia-Antipolis, Directeur de thèse
Mongi BEN GAID
Ingénieur de recherche, IFP Energies nouvelles, Co-Directeur de thèse
Mohamed OULD ABDELLAHI
Chef de département, IFP Energies nouvelles, Invité

Remerciements

Je voudrais remercier toutes les personnes qui m'ont soutenue, de près ou de loin, durant ces trois années de thèse.

Tout d'abord, je voudrais remercier les deux personnes qui m'ont fait confiance depuis le premier jour, Daniel Simon, mon directeur de thèse et Mongi Ben Gaid, mon encadrant IFP Energies nouvelles. J'estime que j'ai eu beaucoup de chance d'être encadrée par eux, autant sur le plan professionnel que personnel.

Daniel, je le remercie pour tous ses conseils et d'être toujours présent. Que ce soit Grenoble ou Montpellier, la distance ne nous a jamais empêchés de bien communiquer ni de travailler efficacement, bien qu'on ait souhaité pouvoir faire marcher la vidéo-conférence!

Mongi, je le remercie pour son aide, son enthousiasme et de toujours trouver le temps pour moi malgré son calendrier hyper chargé. Merci de m'avoir toujours guidé tout en me laissant la liberté de faire mes propres choix.

Je tiens à remercier Pierre Siron et Petter Krus d'avoir accepté d'être rapporteurs de cette thèse. Je remercie également Olivier Séname, Thierry Soriano et Ramine Nikoukhah d'avoir bien voulu faire partie des membres du jury de thèse.

Je remercie également Mohamed Ould Abdellahi, notre chef de département, de m'avoir toujours soutenue et d'avoir cru en moi et en mes travaux. Merci d'être toujours à l'écoute.

Mes travaux de thèse ne seront pas ce qu'ils sont sans l'aide et la collaboration étroite de certaines personnes. Nicolas Pernet, que tout le monde ou presque croyait être mon encadrant, je le remercie pour sa confiance en moi, pour son soutien, son humour, son amitié et d'être toujours aussi serviable. Laurent Duval, je le remercie pour son aide, sa sagesse, ses idées ingénieuses et son amour pour la science.

Je tiens à remercier l'ensemble du département R112 et particulièrement toutes les personnes que j'ai côtoyées tous les jours pendant ces trois années: Bruno, Rodolphe, Manu, Dominique, Arsène, Michel et Fabien. Merci pour leur sens de l'humour, leur accueil et de m'avoir tout de suite mise à l'aise. Bien que je sois restée un long moment l'unique fille et doctorante du département, je ne me suis jamais sentie mise à l'écart.

Je voudrais également remercier nos chères assistantes Carole et Françoise pour leur bonne humeur ainsi que pour leur inestimable aide. Merci aussi à Aurélie, Mai et Camille pour leur joie de vivre, c'est agréable d'avoir un peu de solidarité féminine. Je tiens également à remercier Amel pour son aide et sa gentillesse.

Mes pensées vont aussi à toutes les personnes qui ont partagé mon bureau et à ceux qui ne sont plus à IFP, Hassen, Cyril, Leslie, Hérald et Bertrand, ce fut un plaisir de vous connaître et merci pour toutes les discussions partagées avec vous.

Ces remerciements n'auront pas de valeur sans une très grande pensée à mes chers parents, je tiens à leur dédier ma thèse et à les remercier pour leur soutien malgré la distance qui nous sépare. Une pensée particulière à mon grand frère Leith, ma petite sœur Soussou, ma belle-sœur Héla et ma petite nièce Leila, je voudrais leur témoigner tout mon amour. Je voudrais aussi remercier ma belle-famille pour m'avoir toujours soutenu et encouragé. Du tout profond de mon cœur, je vous souhaite plein de bonheur.

Et pour finir en beauté, je tourne mes pensées vers la personne qui partage ma vie, mon cher époux Zied, je voudrais lui exprimer ma gratitude pour son soutien, son amour, sa patience et ses encouragements, particulièrement dans les moments de doute. Merci d'être toujours à mes côtés et de me donner la force d'avancer.

Contents

List of Figures

List of Tables

Abbreviations

AFR	**A**ir **F**uel equivalence **R**atio
API	**A**pplication **P**rogramming **I**nterface
ARTS	**A**daptive **R**eal **T**ime **S**ynchronization
BDF	**B**ackward **D**ifferentiation **F**ormula
BDC	**B**ottom **D**ead **C**enter
CFD	**C**omputational **F**luid **D**ynamics
CFM	**C**oherent **F**lame **M**odel
CPS	**C**yber **P**hysical **S**ystem
DAE	**D**ifferential **A**lgebraic **E**quation
DAG	**D**irected **A**cyclic **G**raph
DF	**D**irect **F**eedthrough
DEVS	**D**iscrete **EV**ent System **S**pecification
DIRK	**D**iagonally **I**mplicit **R**unge **K**utta
ECFM	**E**xtended **C**oherent **F**lame **M**odel
ECU	**E**lectronic **C**ontrol **U**nit
EDF	**E**arliest **D**eadline **F**irst
EGR	**E**xhaust **G**as **R**ecirculation
ev-MCosim	**e**xtended **v**ersion of **M**odular **Cosim**ulation
FEDEP	**Fe**deration **D**evelopment and **E**xecution **P**rocess
FIFO	**F**irst **I**n **F**irst **O**ut
FH	**F**erris **H**orn
FMI	**F**unctional **M**ock-up **I**nterface
FMU	**F**unctional **M**ock-up **U**nit
FOM	**F**ederation **O**bject **M**odel
GPVS	**G**raph **P**artitioning by **V**ertex **S**eparator
GTE	**G**lobal **T**runcation **E**rror
HIL	**H**ardware-**I**n-the-**L**oop
HLA	**H**igh **L**evel **A**rchitechture
HP	**H**ypergraph **P**artitioning
HPC	**H**igh **P**erformance **C**omputer
IBV	**I**nitial **B**oundary **V**alue
IVP	**I**nitial **V**alue **P**roblem
LLF	**L**east **L**axity **F**irst
LP	**L**ogical **P**rocessor
LSODE	**L**ivermore **S**olver for **O**rdinary **D**ifferential **E**quations
LTE	**L**ocal **T**runcation **E**rror
MCosim	**M**odular **Cosim**ulation
MIL	**M**odel-**I**n-the-**L**oop
MOM	**M**anagement **O**bject **M**odel
MPI	**M**essage **P**assing **I**nterface

NDF	**N**on **D**irect **F**eedthrough
ODE	**O**rdinary **D**ifferential **E**quation
OMT	**O**bject **M**odel **T**emplate
PaToH	**Pa**rtitioning **To**ols for **H**ypergraphs
PCP	**P**riority **C**eiling **P**rotocol
PDE	**P**artial **D**ifferential **E**quation
PFI	**P**ort **F**uel **I**njector
PIP	**P**riority **I**nheritance **P**rotocol
QSS	**Q**uantized **S**tate **S**olver
RCosim	**R**efined **Cosim**ulation
RCP	**R**apid **C**ontrol **P**rototyping
RK	**R**unge **K**utta
RM	**R**ate **M**onotonic
RTI	**R**un **T**ime **I**nfrastructure
SI	**S**park **I**gnition
SIL	**S**oftware-**I**n-the-**L**oop
SOM	**S**imulation **O**bject **M**odel
SRTS	**S**caled **R**eal **T**ime **S**ynchronization
sv-MCosim	**s**tandard **v**ersion of **M**odular **Cosim**ulation
TDC	**T**op **D**ead **C**enter
TLM	**T**ransmission **L**ine **M**odeling
VVT	**V**ariable **V**alve **T**iming
WCET	**W**orst **C**ase **E**xecution **T**ime
WR	**W**aveform **R**elaxation

Part I

Introduction

Chapter 1

Introduction

Foreword

This document synthesizes the three-year PhD thesis, entitled "Distributed real-time simulation of numerical models: application to powertrain", under the supervision of Daniel SIMON[1] *and Mongi BEN GAID*[2]*. It has been financially supported by* IFP Energies nouvelles *and achieved in the Technology, Computer Science and Applied Mathematics division in the continuity of the previous PhD work of Cyril FAURE [1]. The thesis work has resulted in several publications summarized at the end of this manuscript.*

1.1 Context

A major challenge of the 21^{th} century is to successfully achieve 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 price is continually increasing because its availability is expected to vanish during this century. Reducing fuel consumption and diversifying energy sources are major challenges in this field.

On the other hand, global warming and climate change are part of the main concern for global governments, leading to significant considerations to limit pollutant emissions.

In these perspectives, for the automotive industry, regulations are increasingly stringent in terms of fuel consumption and pollutant emissions reduction and new vehicles must comply with these rules in order to be sold. For example, the emission regulation in the European Union[3] aims to reduce harmful exhaust emissions, in particular Nitrogen (NOx) and Particulate Matter (PM) as it is shown in figure 1.1.

These requirements strengthen the need to rapidly adapt new engine concepts, design and related control strategies which implies using multiple technologies that raise the number of actuators to control.

[1] INRIA: `http://www.inria.fr/`

[2] IFP Energies nouvelles: `http://www.ifpenergiesnouvelles.fr/`

[3] `http://europa.eu/legislation_summaries/environment/air_pollution/mi0029_en.htm`

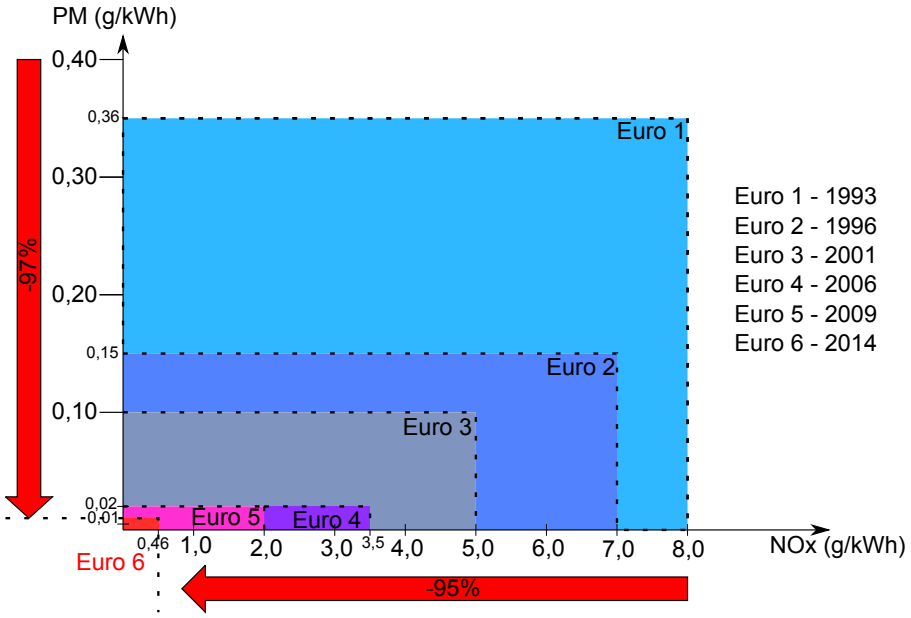

Figure 1.1: Emission regulation Euro 6.

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

1.2 Problem description

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, etc.). The Functional Mock-up Interface (FMI) specification has been proposed to improve this issue [2].

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.

However, the computational power improvement in today processors is mainly driven by the augmentation of the number of cores per processor, rather than in cores frequency increase. To solve this issue, the co-simulation approaches can provide significant improvements by allowing to simulate together models coming from different areas, and to validate both the individual behaviors and their interaction [3]. 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.

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. These interactions between the real components and the simulated components define the Hardware-In-the-Loop (HIL) simulation. The models (simulated components) are intended to validate controllers (real components), e.g., to combine high efficiency with clean combustion. To perform 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 [4].

However, the use of high-fidelity HIL simulation for controls validation is usually prevented by the performance limitation of widely used single-solver/single-core simulation approaches. Besides, simulating such complex systems is time consuming in term of calculations, and reaching real-time is often out of the capabilities of single processors. That is why, parallel computing could be performed, by splitting the models into several sub-models that are concurrently simulated on several processors, to ensure the compliance with the real-time constraints.

The data dependencies, due to the coupled variables between sub-models, lead for waiting periods and idle processor time, decreasing the efficiency of threaded parallelism on the multi-core platform. Therefore, these dependency constraints should be relaxed as far as possible. However, to avoid too large numerical errors in the simulation results, a minimal synchronization between sub-models must be achieved, and the parallelism between computations must be carefully restricted. Hence, the relaxation of the synchronization constraints, while guaranteeing correct simulation results, needs to split the model properly before distribution over several CPUs. An efficient decomposition relies on knowing how and where to cut in order to decouple subsystems as far as possible. Relaxed data dependencies may lead to slack synchronization between sub-models, until reaching an acceptable trade-off between the computation costs and the simulation precision.

1.3 Contributions

This thesis investigates and proposes some analytical and experimental methods towards distributed real-time co-simulation of hybrid dynamical models under slackened synchronization. The term “distributed” refers in this thesis to the distribution of the tasks (or models) over a parallel architecture (multi-core). In fact, the thesis work seeks in particular to define solutions to exploit more efficiently the parallelism provided by multi-core architectures using new methods and paradigms of resource allocation. These solutions aim to validate complex phenomenological models directly through real-time HIL simulation.

The first phase of the thesis studies the possibility of using step-size and order control numerical integration methods with events detection in the context of slackened real-time simulation. It shows the importance of the system splitting, aimed for modular co-simulations, to provide potential speed-ups just by relaxing the number of discontinuities per subsystem. The speed-up refers to how much the proposed (parallel) approach is faster than the corresponding sequential single-thread single-core approach. The speed-up is supra-linear when it is greater that the number of used cores or processors.

Besides, a convergence analysis of the different model of computations used in the context of IFP Energies nouvelles (more precisely in the xMOD tool) shows in a theoretical way that simulation errors are function of the accuracy of the used numerical solver (integration step, order), the models coupling and the communication step. This result confirms and strengthens the followed methodology throughout this thesis.

In addition, we propose two ways to split the models. The first is performed from the physical point of view and it is based on the expertise of the specialists. The second is referring to the structural analysis of the model, based on incidence matrices of states and events, and it is targeting models with no obvious partitioning.

Moreover, the execution order (data dependencies) between loosely coupled models is studied to show the trade-off between the simulation speed and results accuracy. We propose for this aim a new method of co-simulation that allows the full parallelism between the models. It consists in focusing the scheduling only on the inputs/outputs operations, so that it results in supra-linear speed-ups without adding errors related to their execution order. Besides this proposed method eases the engineer work in system splitting, since the required execution order of the data flow is achieved through the scheduling.

Finally, the delay errors due to the communication step-size between the models are improved thanks to a proposed context-based inputs extrapolation. The originality of this method is to bring the concept of contexts to a polynomial prediction, to cope with the stiffness and the discontinuities present in the dynamical systems. Thanks to this new approach, the communication step between the loosely coupled models can be stretched, which allows important simulation speed-ups with an acceptable accuracy of the simulation results.

All the proposed approaches target constructively to enhance the simulation speed for the compliance to real-time constraints while keeping the quality and accuracy of simulation results under control. The validation of these methods is performed through several experiments on complex phenomenological internal combustion engine models. Finally, the proposed methodologies and functionalities are developed in the xMOD tool and expected to be exploited in the next commercial version.

1.4 Outline

This document is composed of four parts where each one is divided into several chapters.

This part gives an overview of the context of the conducted work during this thesis with the different contributions.

Part II covers the state of the art around the different domains of interest of this thesis. First, the different aspects of the validation of complex systems, including the real-time simulation in a systemic approach and the FMI standard for Model Exchange and Co-Simulation (chapter 2). Then, the modeling of complex (hybrid dynamical) systems is formalized and modeling languages and tools are presented (chapter 3). After that, both numerical methods, through time integration and state quantization, are studied (chapter 4) and different approaches of parallelization are investigated in chapter 5. Finally, chapter 6 interfaces the state of the art and the proposed contributions.

Part III exposes the contributions made in the context of IFP Energies nouvelles. First, the case study of an internal combustion engine is introduced (chapter 7) to show the challenging complex systems. This case study will be used for the validation of the different proposed

methods. Then, chapter 8 presents the contribution of the model decomposition on both precision and time computing and shows the advantage of using variable step solvers and root-finding algorithms. Chapter 9 presents the modular co-simulation and a theoretical evaluation of induced errors. Chapter 10 deals with another way of decomposition, based on the relationships between states and events. In chapter 11, a new method that combines a modular co-simulation with a refined multi-core scheduling is exposed. This methods reduces the induced errors due to the co-simulation and eases the system splitting for engineers. Finally chapter 12, presents a new way of extrapolation, based on different contexts, that deals with the hybrid side of dynamical models and improves the accuracy of the simulation results without reducing the speed-up enabled by the parallel execution.

Last, Part VI concludes this thesis by summarizing the work and discussing several perspectives.

Part II

Context and problem position

Foreword

This part presents the state of the art around the different domains of interest of this thesis. It is made up with a critical point of view to position the thesis in comparison with the literature. First, different aspects of the simulation of complex systems are emphasized, in particular with the presentation of the real-time simulation framework. Then, the modeling of complex (hybrid dynamical) systems is formalized, to be further used by the different methods of physical systems' resolution. Both numerical methods, through time integration and state quantization, are examined for this purpose. Finally, different approaches for models and solvers parallelization are investigated.

Chapter 2

Complex systems validation through real-time simulation

2.1 Introduction

This chapter describes the different stages of the systemic validation of complex systems, from the system simulation to the real-time simulation. It exposes the different involved areas, as the systemic approach, the real-time computing and the standards for co-simulation.

2.2 Systemic approach

Complex systems such powertrains require the design of both multi-disciplinary physical models (mechanical, electrical, thermodynamic and chemical) and computational components (hierarchy of controllers).

Such highly complex systems are composed of a large diversity of elements linked together by strong interactions and need a systemic approach for the design and validation phases. In fact, the systemic approach to problems assumes that systems are seen as a whole, which means that their parts are not seen individually [5]. Moreover, the focus is just on the total inputs and total outputs of the system, without worrying on which part of the inputs goes to which subsystem [6].

Such an approach is concerned by the total system performance. This means that the consequence of any changes in some parts of the whole system is not as important in it-self than the consequence of the interaction of these changes. The reason is that there are some system's properties that can only be analyzed in an appropriate way from a holistic point of view.

With the systemic approach, all involved engineering fields start design phase at the same time and refine their models further as they advanced. The coupling of different domain models does not require the knowledge of all the specialties on a very fine scale because the validation is achieved at a system level.

2.3 System simulation

2.3.1 Numerical simulation

For complex systems, it is often difficult, indeed impossible, to derive their analytical solutions. This is why the numerical simulation is used at an early stage for designing equipment, setting systems such as feedback loops and analyzing dynamic phenomena.

The main purpose of numerical simulation is to approximate the behavior of the complex physical phenomenon as faithfully as possible and to obtain the most possible accurate results. This means that the only focus is on the precision of the simulation results and the computational time is out (or at a low level) of concerns.

2.3.2 Co-simulation for Model-In-the-Loop

Co-simulation means the combined use of different heterogeneous simulators. The coupling or integration of these simulators will be the basis for the simulation of the entire system. Within the framework of complex systems simulation, multiple specialties are involved, where each of them requires that the modeling and the design of its own simulator have to be performed with their own specific tool and language. The numerical resolution of each model could be done in its corresponding simulation tool or directly in the final simulation tool where all models are integrated together.

When a control law component is included with the other models, we are talking about the Model-In-the-Loop (MIL) simulation (see figure 2.1).

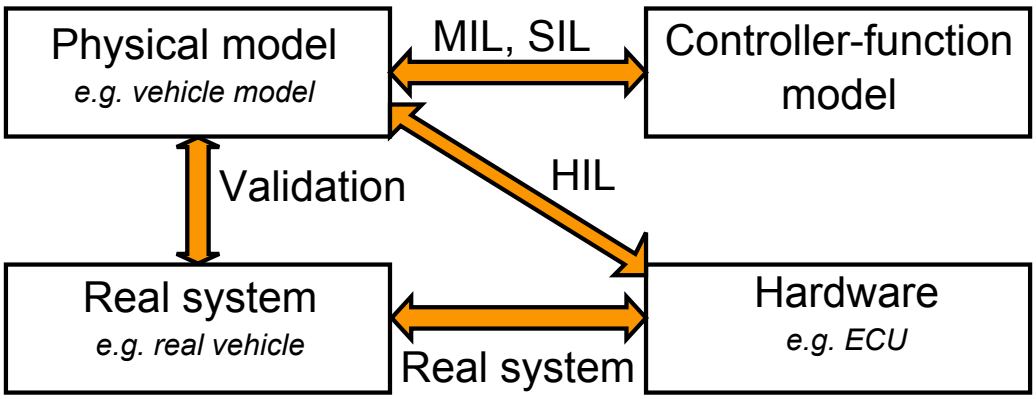

Figure 2.1: The different steps of simulation for system validation.

Co-simulation requires the synchronization of involved models execution by the integrating environment. It means that each model must be able to detect, locate and respond in time events sent by the other model, which is not trivial at all. In fact many work deal with synchronization algorithms between simulators [7]. Indeed, MIL co-simulation allows only for the prototyping and the validation of the system in a virtual way by looping and correcting the models until verifying an acceptable behavior of the system.

To allow the import of coupled models or simulators in an easy way, the idea is to impose on the modeling and simulation tools the respect of a same specified model interface.

2.4 Real-time simulation

Real-time simulation implies matching two concepts of time:

- Real-time: is the physical time or the time reference of the real physical system that we want to model and simulate;
- Simulated time: is the elapsed time during the execution of the simulation that can be measured by the integrator clock.

In other words, in real-time simulation, the simulation time needs to be meshed to the real-time.

2.4.1 Software-In-the-Loop simulation

As for MIL, the Software-In-the-Loop (SIL) phase (see figure 2.1) considers only simulated elements. However, it takes into account the controller implementation code (that respects target execution hardware constraints, such as fixed point calculations and memory limitation). In the case of real-time SIL simulation, data exchanges between simulators and controllers have to be at the same rates as the real components. SIL validates the correct behavior of control softwares.

For real-time SIL simulation, the data from each model are synchronized. That is, data for a given time must be processed together and should not be mixed with other data from a different time. In [8], real-time constraints are described through the following example: for a system made up of a controller and a plant modeled by two separated PCs, in each sampling interval, data transmission and the computation of control algorithms in the controller PC must be completed as well as data transmission and the computation of plant dynamics in the plant PC.

SIL has a lower cost comparing to Hardware-In-the-Loop (HIL) and Rapid Control Prototyping (RCP) because it does not need any hardware that could be relatively expensive. Indeed, for HIL, the controller model is replaced by a real hardware whereas for RCP, the plant is substituted by the real physical system.

2.4.2 Hardware-In-the-Loop simulation

After validating the real software in a real-time SIL simulation, the real hardware can be checked in a real-time Hardware-In-the-Loop (HIL) simulation (see figure 2.1). HIL simulation consists of a combination of simulated and real components, which means that a real component can be replaced by an artificial one.

Industrially, it is often used to test embedded software on its final execution platform due to the unavailability of some parts of the real system. It offers many advantages [9]: reducing costs by requiring only some equipment, rapid prototyping, good representation of the system, flexibility by allowing repetitive tests and trying destructive tests without impacting on the hardware.

The use of HIL in the automotive field has been introduced to validate the ECUs (Electronic Control Units) that will be embedded in the vehicle [10]. HIL approach aims to go as far as possible in the realism, while remaining in the software world, and to respect the real dynamics of the physical system to be controlled (vehicle, driver, engine, etc.).

The simulated model (physical system to be controlled) should work ideally with the real system dynamics. This means that if in reality a valve in the physical system takes x microseconds to open, in the simulated model, the calculation that reproduces this opening must be exactly provided x microseconds.

In HIL simulation, the computation time seen by the computer is strictly equal to the measured time by the man in the real world, which means that the computer will be "deceived" by a completely artificial environment. Indeed, as it does only "see" the outside world through the electrical signals which it receives or sends, simulated sensors and actuators can deceive it by delivering the same forms and rhythms of signals.

2.5 Real-time computing

2.5.1 Real-time systems

Real-time systems are often modeled by concurrent tasks, where each task is a process of a computational activity. The term real-time does not mean actually "as fast as possible" computing but it implies the existence of real-time constraints that have to be met. These constraints are often represented by the deadlines of tasks.

The real-time system can be presented by a simple architecture as in figure 2.2, where the interaction between the controlled system and the controller are triggered by real-time constraints.

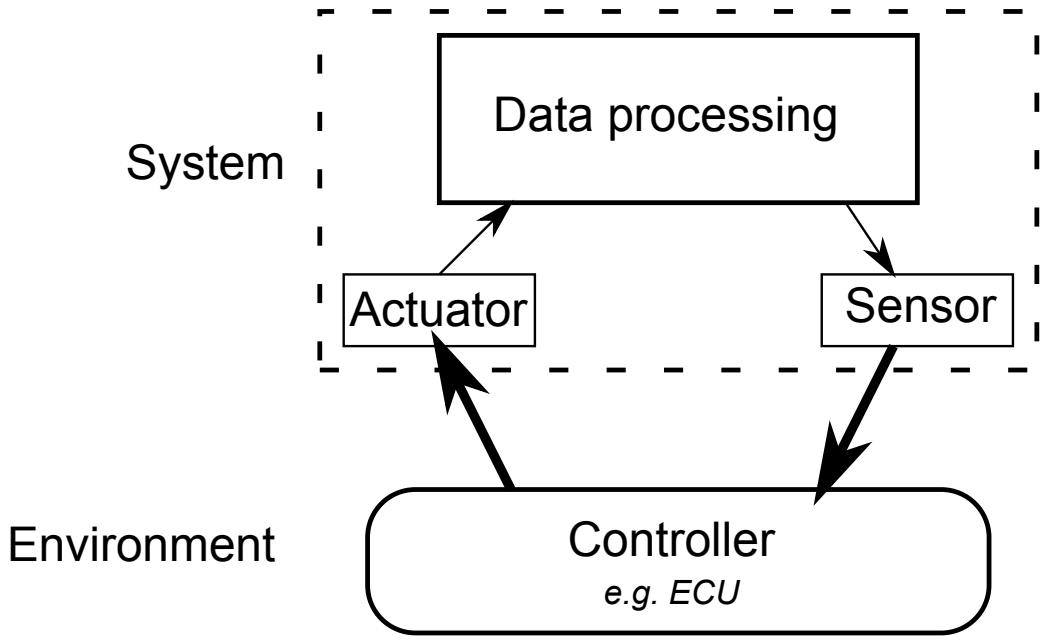

Figure 2.2: Real-time control system.

2.5.2 Real-time constraints

"Overruns are defined as situations, where, in spite of our best efforts, the simulation engine is unable to perform all of the required computations in time to advance its state to the next clock time, before the real-time clock interrupt is received" [11].

According to the kind of involved constraint, failing to meet a deadline (overruns) may involve several consequences.

Hard real-time

The constraints are considered as "hard" when it is mandatory that the system satisfies the deadlines. Missing such constraints implies the system's failure.

Critical hard real-time

The critical constraints are found in critical embedded systems [12], where the failure of the system leads to dramatic consequences as serious injury to people, severe damage to equipment or catastrophic harm to environment.

Soft real-time

The constraints are considered "soft" if not meeting a real-time constraint degrades the system performance but without undermining the desired behavior. For example, the live video streaming is a soft real-time system where the loading time may exceed the real-time. The system here will not be damaged due to the excess of time but the expectation of the user (viewer) will not be satisfied.

Men-In-the-Loop simulators may be considered as soft real-time systems where deadlines can be missed in a non-predictable way.

Firm or Weakly hard real-time

In our point of view, real-time simulators for HIL validation may be considered as weakly-hard real-time systems [13]. Unlike soft real-time system, the weakly hard real-time system may miss a specified number of deadlines, but in a predictable way, in order to guarantee a level of quality of service.

The missed deadline may occur but it is controlled and its consequences are expected. Most real-time simulations specify the maximum percentage of overruns as e.g. 1% or 2%.

In fact, [14] showed that control feedback loops on an aircraft model are robust enough to slacken the hard real-time approach and tolerate a specified number of missed deadlines. The performance and stability of the system are kept while using more flexible and fault-tolerant systems.

In this thesis, we consider weakly hard real-time HIL simulation and focus on computation time aspects, where small enough computation times are necessary to comply with real-time constraints. In fact, the failures of the system are not destructive since the system is a simulator. Real-time constraints are needed to be firm for results correctness.

2.5.3 Real-time scheduling

Real-time scheduling defines the execution order of each task (or process) on the processor.

Real-time task parameters

To perform the scheduling, each task T_i is characterized by several parameters:

- Release time r_i: or activation date, is the time at which a task is ready to be launched;
- Start time s_i: is the time at which a task started its execution;
- Computation time C_i: is the execution time of a task without interruption, it is usually taken as the Worst Case Execution Time (WCET);
- Finishing time f_i: is the time at which a task completed its execution;
- Response time R_i: is the elapsed time between the release time and the finishing time of a task. Its value is determined by the following relationship $R_i = f_i - r_i$;
- Absolute deadline d_i: is the time at which a task must be finished;
- Relative deadline D_i: is the deadline relative to the release time. Its value is determined by the following relationship $D_i = d_i - r_i$.

All these parameters are illustrated in figure 2.3.

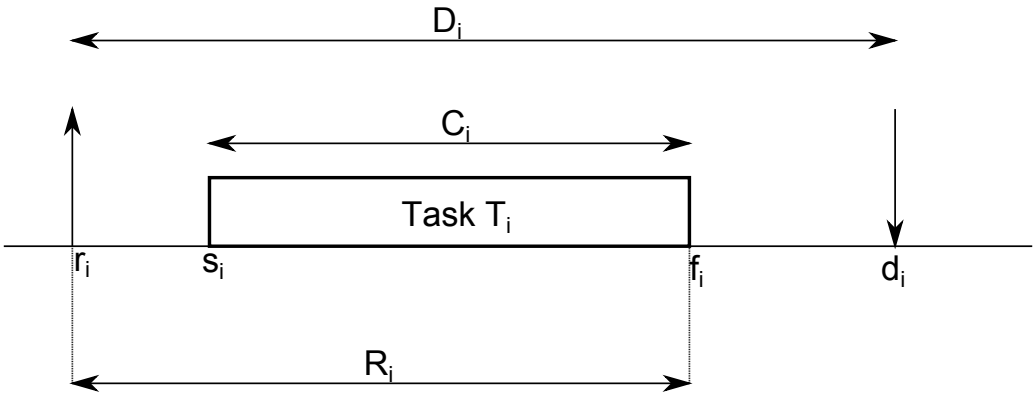

Figure 2.3: Parameters of a real-time task.

A task can also be:

- Periodic, if there is an infinite sequence of instances (or jobs) as $r_{i,j} = r_{i,j-1} + P_i$, where P_i is the period;
- Aperiodic, if there is no periodicity;
- Sporadic, if there is a minimum time between activation dates $r_{i,j} \geq r_{i,j-1} + k_i$, where k_i is the pseudo-period.

Real-time scheduling

To ensure that all timing constraints of a set of tasks will be met or not beforehand, the schedulability analysis is performed. It consists in verifying that the total processor utilization factor U induced by the set of tasks $(i = 1, \ldots, N)$ is under the "schedulability least upper bound" U_{lub} [15] of the used scheduling algorithm:

$$U = \sum_{i=1}^{N} \frac{C_i}{P_i} \leq U_{\text{lub}}.$$

A scheduling is called feasible, if all tasks are executed in accordance with the specified constraints. A set of tasks is schedulable if at least one scheduling algorithm is able to produce a feasible schedule. A scheduling algorithm is optimal when it can schedule all task sets that other algorithms cannot.

A scheduler can be either off-line or on-line. It depends if the scheduling decisions are made prior to or during the running of the system. For off-line algorithms, the tasks have to be periodic. A scheduler may also be either preemptive (e.g. Least Laxity First (LLF)) or non-preemptive (e.g. First In First Out (FIFO)). The preemption consists in suspending the execution of a task, because a higher priority task becomes ready to run, then in resuming it later without affecting its behavior. Finally, a scheduler may be with a fixed priority (e.g. Rate Monotonic (RM)) or with a dynamic priority (e.g. Earliest Deadline First (EDF) [15]).

Most real-time systems present a mixture of periodic and aperiodic tasks, several number of bandwidth preserving algorithms have been proposed for this issue as the Priority Exchange Server, the Deferrable Server and the Sporadic Server [16]. Besides, for more realistic real-time systems, tasks interact for synchronization, mutual exclusion protection of a non-sharable resource and precedence relations through for example semaphores. Several protocols were proposed to bound and reduce the blocking, as the Priority Inheritance Protocol (PIP) and the Priority Ceiling Protocol (PCP) [17].

Multi-core scheduling

In addition to the temporal execution of the set of tasks, for multi-core scheduling, the distributed execution has to be solved too. Consequently, usual and even optimal scheduling for mono-processor cannot be applied for multi-processor scheduling [18], because the distributed execution adds another complexity to the tasks' priority assignment problem.

In multi-core scheduling, there are two important categories [18]:

- Partitioning is a method that assigns each task to a specific processor before the system runs. Then, a usual mono-processor scheduling algorithm can be used on each processor. The partitioning offers the simplicity and the reuse of the mono-processor scheduling techniques, however at the cost of the utilization bound limitation [19];
- Global strategy is a method that allows the tasks' migration from one processor to another, known as restricted migration. It also allows the jobs' migration, known as full migration, by making possible to resume the jobs' execution on another processor. Several global scheduling algorithms can achieve a utilization bound of 100%, however at the cost of overheads.

Several work on semi-partitioning approaches aim to combine the advantages of each approach, low dispatching overheads for the partitioning and efficient use of the processors for the global strategy. For example, [20] formulated a new EDF-based semi-partitioned scheduling algorithm, named HIME, with an utilization bound of 75% by limiting the number of tasks' migrations to $m/2$ and jobs' migrations to $m-1$, with m the number of processors.

Besides, several protocols were proposed to bound and reduce the blocking in the case of distributed scheduling. For example, Flexible Multiprocessor Locking Protocol [21] provides the information about the duration of locking shared resources, for both partitioned and global schedulings.

Finally, heuristics can be used to schedule real large size task sets. A heuristic algorithm,

defined in [22], is used to optimize the distribution and scheduling of the tasks onto a multi-processor architecture. In chapter 11, a similar scheduling heuristic method is used to improve the speed and the accuracy of the co-simulation.

2.6 Standards for co-simulation

It is clear that simulation is a crucial step of validation. To capitalize on the existing simulation effort and to facilitate the co-simulation of coupling simulators that may be running on a distributed architecture, standards exist to support interoperability, exchange and reusability of simulations. These standards can prevent tedious development for products already compatible and facilitate the components' replacement. Functional Mock-up Interface (FMI) and High-Level Architecture (HLA) are part of these standards that enables modelers to reuse already developed and tested simulations as components.

2.6.1 Functional Mock-up Interface FMI

The FMI specification [2,23] is a tool independent and open standard[1] designed in the ITEA2[2] MODELISAR project and continued at its end by the Modelica Association as a MAP[3].

The overall goal of FMI is to support both the exchange and the co-simulation of dynamic models (as CPS) by combining software components provided by different sources (see figure 2.4) for MIL, SIL and HIL simulation and for embedded systems. In particular, it was intended from the beginning to support the use of the AUTOSAR[4] standard and of the Modelica language[5].

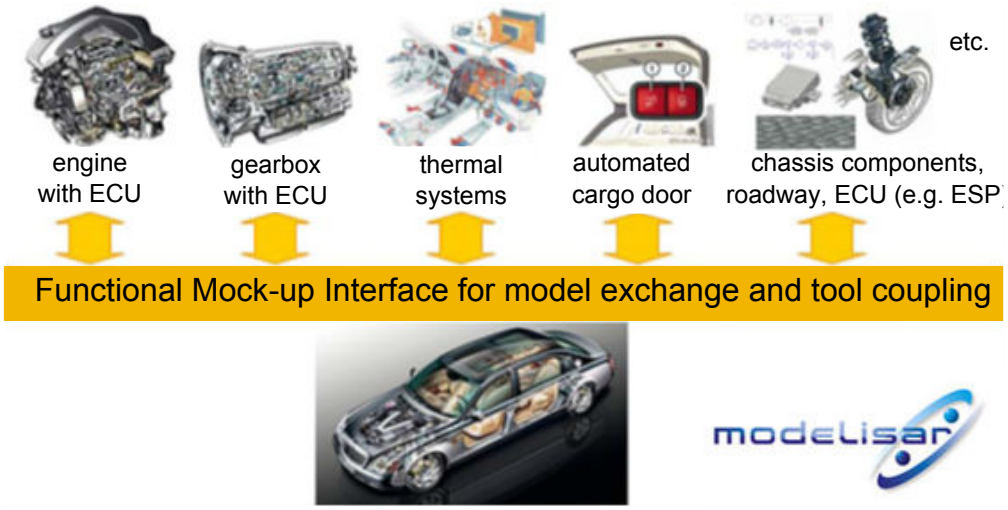

Figure 2.4: Modelisar vision of automotive system co-simulation.

The simulators may be exported, from original authoring tools that support the FMI, as Functional Mock-up Units (FMUs) and then imported in a co-simulation environment. Hence, they

[1] https://www.fmi-standard.org/
[2] Information Technology for European Advancement
[3] Modelica Association Project
[4] http://www.autosar.org/
[5] https://www.modelica.org/documents

cooperate at run-time thanks to the FMI definitions of their interfaces, and to the master algorithms of these environments.

The FMU is a zip file (*.fmu) that contains different component:

- *modelDescription.xml*: A model description scheme represented by an XML file. It contains the description of the different data flow between the FMU and the simulation tools (inputs, parameters, outputs, continuous states, discrete states, event indicators, etc.);
- *FMI functions*: They are standardized C-functions that can be available in two forms:

  - *Binaries*: Optional directory depending on the platform (e.g. win32, win64, linux32). It contains shared libraries (e.g. DLL, lib) which contains the implementation of the FMI functions (standardized C-functions);

  - *Sources*: Optional directory containing all the C sources of the model interface. The FMI functions are used by the tool to create one or more instances of the FMU and to run them together with other models;
- *Documentation*: Optional data, images and documentation of the model.

The FMU may either be self-integrating (FMI for Co-Simulation) or require the numerical integration performed by the importing tool (FMI for Model Exchange).

FMI for Model Exchange

FMI for Model Exchange specification, illustrated in figure 2.5, provides the encapsulation of models equations in well-defined components and interfaces and it allows for solving independently the sub-models using custom solvers. The access on model equation is performed through function calls such as "*fmiGetDerivatives*" to return the value of the state derivatives and "*fmiSetContinuousState*" to set a new value to the continuous state vector.

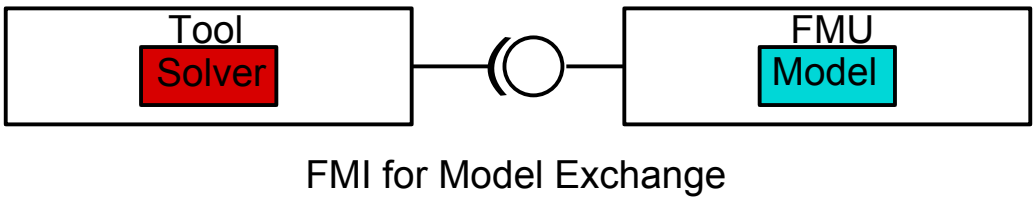

Figure 2.5: Main design idea of FMI for Model Exchange.

FMI for Co-Simulation

FMI for Co-Simulation specification, illustrated in figure 2.6, provides interfaces between master and slaves. It allows for the coupling of several models together with their solvers in a co-simulation environment, designed to manage the data exchange and synchronization between subsystems. XML model description provides information about the slaves, especially a set of capability flags. They characterize the ability for a slave to support advanced master algorithms such as the use of variable communication step-sizes "*canHandleVariableCommunicationStepSize*" and higher order signal extrapolation "*canInterpolateInputs*".

In this thesis, we are especially interested on the FMI for Model Exchange because it allows us to operate internally on numerical solvers. More details can be found in chapter 8.

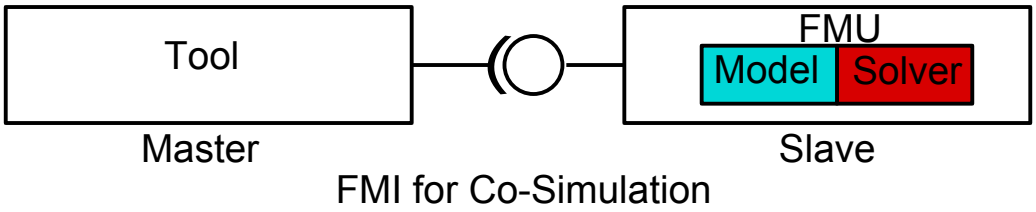

Figure 2.6: Main design idea of FMI for Co-Simulation.

2.6.2 High-Level Architecture HLA

The HLA is an IEEE standard [24], originally developed by the U.S. Modeling and Simulation Coordination Office (M&S CO)[6] to facilitate distributed environments, suitable for military simulations. Nowadays, it is increasingly widespread in the civil sector.

The distributed simulation is called a "federation" that comprises several components known as "federates". This specification keeps the simulation components usable even when the data model, called Federation Object Model (FOM), is changed. The federates are regulated through the Run Time Infrastructure (RTI), which is the central point for communication and data exchange.

HLA standard comprises four elements:

- Ten rules defined by the HLA standard concerning the global functioning of the federations and the federates;
- An Object Model Template (OMT) that provides standardized documentation of the HLA object models: the Federation Object Model (FOM), the Simulation Object Model (SOM) and the Management Object Model (MOM);
- An API (Application Programming Interface) specification that provides a set of low-level functions that uses services of the standard HLA provided by the RTI;
- A number of recommendations for the design and development of HLA federations: the FEDEP (Federation Development and Execution Process).

Several implementations of the RTI (commercial and open source) were developed depending on the HLA standard version, the programming language and the platform. ONERA was one of the first organizations to achieve an open source RTI for HLA, named CERTI [25] and several work of distributed simulation for real-time systems were performed in [26–28]. In [29], a proposed approach allowed the communication (based on HLA) between OpenModelica and OpenMASK[7] simulators in the context of real-time simulation. The synchronization of the simulators was based on a global simulation time advancement. The OpenMASK was triggered by OpenModelica which is synchronized with the clock time. Figure 2.7 describes the global architecture of a HLA simulation.

Most important services of the RTI are:

- Time management regulates the advancement of the federates' time;
- Event management passes the messages between federates in the form of events;
- Object management concerns the propagation of the update of objects and attributes;

[6] `http://www.msco.mil/`
[7] `http://www.openmask.org/`

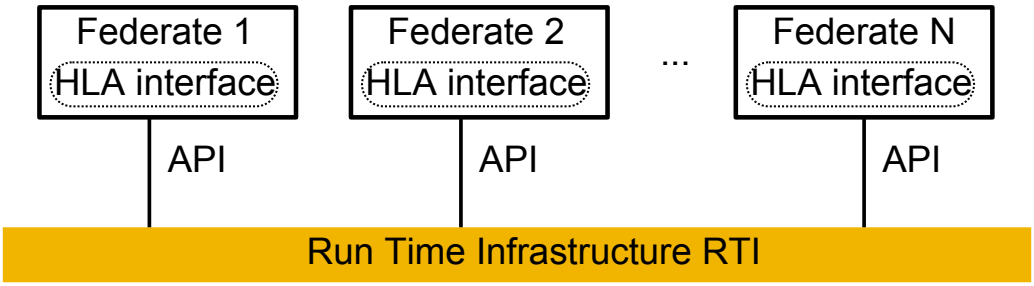

Figure 2.7: HLA Federation.

- Declaration management deals with the publishing and subscription of objects and attributes;
- Ownership management avoids conflicting updates (an attribute must belong to only one federate).

Recently, several work are interested to combine the use of the HLA and the FMI. A framework is proposed in [30] to use them together by considering the HLA as a master to the FMUs in order to create an entirely generic and stand-alone master for the FMI. This approach makes the FMUs usable as plug and play components on different distributed environments.

2.7 Conclusion

Real-time HIL simulation is required to achieve the validation of complex physical systems. For this aim, before choosing the appropriate solver to integrate and simulate, the system is firstly needed to be modeled. The next chapter describes the formalization of the problem using differential equations and modeling languages.

Chapter 3

Modeling of physical systems

3.1 Introduction to modeling

A model is a simplified representation that can reproduce appropriately a system (or a part of system) behavior. A model can be mathematical, described by differential equations, where it is commonly used in continuous state systems as fluid mechanics, geology, meteorology, etc. It can also be based on learning methods, as artificial neural networks, that is commonly used in signal processing, process control, data classification, etc. Finally, a model can also be both formal (languages with a mathematical definition) and semi-graphical, commonly used in discrete event systems such as logistic processes (organization of services, transportation systems, etc.) and technical processes (telecommunication networks, computer networks, production lines, etc.).

In this thesis, we are especially interested on the first category of modeling for continuous state systems with an additional aspect regarding a discontinuous behavior (localized discrete events) that will be detailed through the differential equations formalism. For discrete event systems (systems with a finite number of changes in a finite interval of time), the modeling can be performed by for example Petri Nets, State Charts, Event Graphs, etc. Note this kind of models can be seen as particular cases of DEVS[1] models (detailed in section 4.3) since it handles discrete event systems, continuous state systems and hybrid systems.

3.2 Basics on differential equations

The evolution of a dynamical system is governed by one or several differential equations. To solve the system, the problem should be well-formulated before choosing the appropriate solver to integrate it. A well-defined Initial Value Problem (IVP) is characterized by:

- a list of quantities to be integrated according to a list of parameters,
- a differential equation for each quantity,
- and initial conditions (or boundary conditions).

Modeling a system can be performed using different kind of differential equations. For example, in the domain of thermodynamics, physics-based models are well-suited for design but their computation is too slow for system simulation and control. They involve usually Partial Differential Equations (PDEs). Semi-empirical models, however, are suitable for dynamics and fast

[1] Discrete Event System Specification

simulation but they have limited insight into physical behavior. They involve usually ODEs and DAEs.

3.2.1 Ordinary Differential Equations

ODEs are the simplest form of differential equations where the unknown functions depend only on a single parameter which is usually the time t. We are interested in this thesis on this particular form.

In that case, an equation with an order greater than one can always be reduced to a set of 1^{st} order equations, which is simpler to solve and get the state space form:

$$\dot{\boldsymbol{X}} = \boldsymbol{f}(t, \boldsymbol{X}), \tag{3.1}$$

with $\boldsymbol{X} \in \mathbb{R}^{n_X}$ is a vector of n_X quantities: dependent variables of interest (state variables), $\dot{\boldsymbol{X}} = \frac{d\boldsymbol{X}}{dt}$ is the time-derivative and $t \in \mathbb{R}_+$ is the time.

Example

The fall of a punctual body being only subjected to the gravity g, is represented by the differential equation of 2^{nd} order:

$$a_z = \frac{d^2 z}{dt^2} = -g,$$

where a_z is the acceleration. The quantities to be integrated are the position z and the velocity v_z, according to the time t (the parameter). The initial conditions are defined by the initial position z_0 and the initial speed v_{z_0}. The problem can be rewritten in a set of two 1^{st} order equations and get the state space form (3.1) with

$$\boldsymbol{X}(t) = \begin{pmatrix} z(t) \\ v_z(t) \end{pmatrix}, \quad \boldsymbol{f}(t, \boldsymbol{X}) = \begin{pmatrix} v_z(t) \\ -g \end{pmatrix}.$$

3.2.2 Partial Differential Equations

PDEs are differential equations where the unknown functions are depending on multiple parameters and the equation involves its partial derivatives. PDEs are used in domains such as thermodynamics, fluid mechanics and acoustics.

As for ODEs, the problem can be rewritten in a set of 1^{st} order equations. For example for a three spatial dimension, the parameters are then the spatial coordinates (x, y, z) and the time t. The PDEs are written in the form

$$\boldsymbol{F}\left(t, \boldsymbol{X}, \frac{\partial \boldsymbol{X}}{\partial t}, \frac{\partial \boldsymbol{X}}{\partial x}, \frac{\partial \boldsymbol{X}}{\partial y}, \frac{\partial \boldsymbol{X}}{\partial z}\right) = 0.$$

Modeling system through PDEs are usually used at early design stage to describe fine-grain dynamics. Besides, solving these kind of equations requires methods that generally imply high computational loads.

3.2.3 Differential Algebraic Equations

DAEs are differential equations involving differential variables similar to ODEs but also algebraic variables given in the implicit form

$$\boldsymbol{F}(t, \boldsymbol{X}, \dot{\boldsymbol{X}}) = 0. \tag{3.2}$$

In other words, a system of equations is called DAE when the Jacobian $\frac{\partial \boldsymbol{F}}{\partial \dot{\boldsymbol{X}}}$ is singular (non-invertible). Unlike differential variables, the derivatives of the algebraic variables are not given explicitly. Note that ODEs (3.1) can be easily rewritten in the form of DAEs (3.2). They are said DAEs with an index zero.

When it is possible to distinguish and separate the algebraic variables from the others (pair $(\boldsymbol{X}, \boldsymbol{X}_a)$ of vectors with $\boldsymbol{X}_a \in \mathbb{R}^{n_{X_a}}$ is the algebraic vector), the problem can be rewritten in the form

$$\begin{aligned} \dot{\boldsymbol{X}} &= \boldsymbol{f}(t, \boldsymbol{X}, \boldsymbol{X}_a), \\ \boldsymbol{0} &= \boldsymbol{\varphi}(t, \boldsymbol{X}, \boldsymbol{X}_a). \end{aligned} \tag{3.3}$$

A DAE written in the form (3.3) is called semi-explicit of index one. The Jacobian of $\boldsymbol{\varphi}$ with respect to $\boldsymbol{X}_a$, $\frac{\partial \boldsymbol{\varphi}}{\partial \boldsymbol{X}_a}$, is then non-singular and the differentiation is only needed when the calculation of $\dot{\boldsymbol{X}}_a$ is required.

DAEs with an index zero or one can be solved easily with standard numerical ODE methods. For high index DAEs, techniques of index reduction are performed, such as Pantelides algorithm [31], structural matrix algorithm [32] [33] and dummy variable substitution.

Dymola[2] solves the problem of handling high index DAE problems and selection of states in an efficient and reliable way. The index reduction procedure consists of two major steps. First, the differentiated index 1 problem is derived (using Pantelides) and then it is used for selection of dummy derivatives [34].

Pantelides algorithm is a symbolic index reduction algorithm that removes structural singularities from a model. When a constraint equation is found, the algorithm adds the differentiated constraint equation to the set of equations. Then, it re-equalizes the number of equations and unknowns, by eliminating one of the integrators that is associated with the constraint equation.

3.3 Problem formalization

3.3.1 Continuous dynamical systems

The dynamical system Σ is the modeling of the physical part of a CPS in the continuous-time domain. It is intended to interact with controllers through inputs and outputs. The controllers represent computational components that are modeled on the discrete-time domain and sampling is the interaction of time-driven and events-driven dynamics of the system.

[2] `http://www.3ds.com/products-services/catia/capabilities/systems-engineering/modelica-systems-simulation/dymola`

Therefore, the continuous state evolution of Σ is governed by

$$\dot{\boldsymbol{X}} = \boldsymbol{f}(t, \boldsymbol{X}, \boldsymbol{U}_{ext}), \tag{3.4a}$$
$$\boldsymbol{Y}_{ext} = \boldsymbol{g}(t, \boldsymbol{X}, \boldsymbol{U}_{ext}), \tag{3.4b}$$

where $\boldsymbol{X} \in \mathbb{R}^{n_X}$ is the continuous state vector, $\boldsymbol{U}_{ext} \in \mathbb{R}^{n_{U_{ext}}}$ the external input vector, $\boldsymbol{Y}_{ext} \in \mathbb{R}^{n_{Y_{ext}}}$ is the external output vector and $t \in \mathbb{R}^+$ is the time.

We assume that Σ is well-posed in the sense that the differential equations are Lipschitz continuous, meaning that a unique solution exists for each admissible initial conditions $\boldsymbol{X}(t_0)$ and consequently $\boldsymbol{X}$ and $\boldsymbol{Y}_{ext}$ are continuous functions. A function $\boldsymbol{f}$ is said Lipschitz continuous if $\exists L \geq 0$ such that $\forall \boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^n$, $|\boldsymbol{f}(\boldsymbol{b}) - \boldsymbol{f}(\boldsymbol{a})| \leq L|\boldsymbol{b} - \boldsymbol{a}|$.

3.3.2 Hybrid dynamical systems

The kind of models, that we are especially interested in, is hybrid systems where the system modeling uses hybrid ODEs. The hybrid side is due to the presence of some discontinuous behaviors, that correspond to events triggered-off when a given threshold called *zero-crossing* is crossed.

Let us provide a formal model, considering a hybrid dynamic system Σ' whose continuous state evolution is governed by

$$\dot{\boldsymbol{X}} = \boldsymbol{f}(t, \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{U}_{ext}) \quad \text{for } t_n \leq t < t_{n+1}, \tag{3.5a}$$
$$\boldsymbol{Y}_{ext} = \boldsymbol{g}(t, \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{U}_{ext}), \tag{3.5b}$$

where $\boldsymbol{X} \in \mathbb{R}^{n_X}$ is the continuous state vector, $\boldsymbol{D} \in \mathbb{R}^{n_D}$ is the discrete state vector, $\boldsymbol{U}_{ext} \in \mathbb{R}^{n_{U_{ext}}}$ the external input vector, $\boldsymbol{Y}_{ext} \in \mathbb{R}^{n_{Y_{ext}}}$ is the external output vector and $t \in \mathbb{R}^+$ is the time.

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

$$\boldsymbol{h}(t, \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{U}_{ext}) = 0. \tag{3.6}$$

The function $\boldsymbol{h}$ is usually called *zero-crossing* function or *event indicator*, used for *event detection* and *location* [35].

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

$$\boldsymbol{X}(t_n) = \boldsymbol{I}(t_n, \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{U}_{ext}), \tag{3.7}$$

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

$$\boldsymbol{D}(t_n) = \boldsymbol{J}(t_{n-1}, \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{U}_{ext}). \tag{3.8}$$

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

This hybrid system model is adopted by several modeling and simulation environments and is underlying the FMI specification [2].

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

Discontinuities are classified according to how they may occur:

- *Time event* is an event that can be known beforehand at a predefined time instant t_n. This time instant is defined at the previous event instant t_{n-1} either by the model or by the model's environment. Periodic sampling events are examples of time events;
- *State event*, it is the *zero-crossing* detection as defined earlier. It is the root of the *event indicator* (3.6);
- *Step event* is an event that occurs after a successful integration step. Step events can be used to dynamically change the continuous states of a model in (3.7), when the previous states are no longer suited numerically.

In hybrid systems, the zero-crossing function $\boldsymbol{h}$ is a set of conditional statements that must be evaluated at every time instant t_n during the numerical integration. That is why, this kind of systems requires a specific solver with obviously a step-size control and especially a root-finding capability to insure the assumption validity of the piece-wise continuity between two consecutive time instant.

In [36], another approach handles the problem of zero-crossing by developing a dedicated type system and causality analysis that ensure the alignment of all discrete changes with zero-crossing events. This kind of approach allows for avoiding discontinuities occurrence during integration by using a programming language for modeling called Zélus, that mix discrete logical time and continuous time behavior and that is derived from synchronous languages.

3.4 Modeling languages and tools

Many modeling languages (e.g. Modelica, SysML, VHDL, Simulink[3], Scicos[4]) were developed to model complex systems that cover multiple application domains at a high level of abstraction through reusable model components.

To build mathematical models of complex systems, it is more practical to use the modular modeling by assembling the models of the different constituent system parts. In this perspective, a complementary classification by causality (causal or acausal) can be added to the modeling classification (physics-based or semi-empirical).

3.4.1 Causal approach

Causal or block-oriented modeling is an imperative (procedural) approach that is similar to the solution algorithm. It describes what the model should accomplish. The model is described via inputs and outputs variables. The equations that describe the system's physics must be in the form where the direction (causality) of signal flow is explicit.

The signal flow representation makes hard to understand the original physical models. Besides, the system decomposition does not correspond to the natural physical structure. In addition,

[3] `http://www.mathworks.com/products/simulink/`
[4] `http://www.scicos.org/`

the need to explicitly specify the causality prevents components reuse in other context, i.e. small changes in the model requires the redesign of the whole model. For example computing current from voltage instead of voltage from current (see figure 3.1).

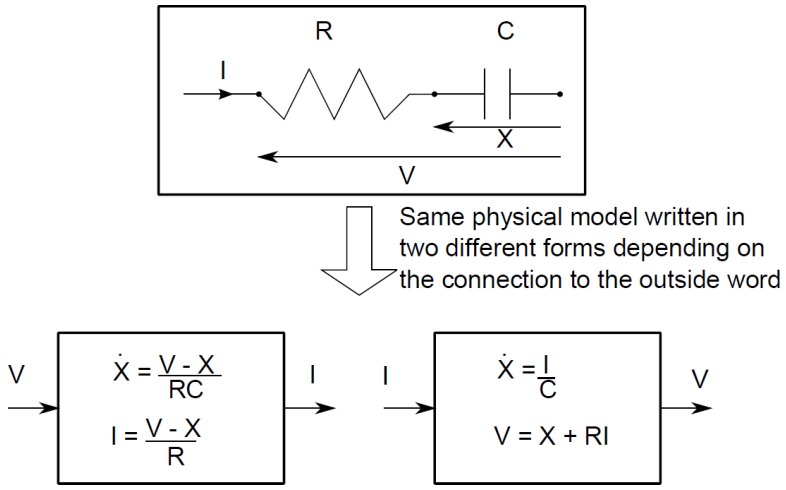

Figure 3.1: Causal (imperative) models.

The main advantages of the block diagram modeling language is that it is straightforward to solve the equations of a model, because it is usually described in the state-space form. Moreover, it is well-suited for control systems since they are signal-oriented rather than physical. Such causal modeling languages can be found in Ptolemy II [37], Simulink, Scicos [38] and AMESIim[5].

3.4.2 Acausal approach

Acausal modeling is a declarative approach that does not carry about the current solution algorithm. It describes what the model should accomplish by equations in context-independent form.

It is based on physical interaction formalized by the connection equations without specifying the causality (see figure 3.2). The appropriate causality constraints are then inferred using both symbolic and numerical methods depending on how the model is being used. Unlike causal languages, acausal languages make possible the re-usability of basic models and the readability of the models. However, it is more difficult to numerically solve the mathematical model because it is not oriented to the solution algorithm. As causal modeling, the acausal modeling support the object-oriented approach too. Modelica is a popular acausal modeling language that can be found in various tools such as Dymola, MathModelica[6], SimulationX[7], MapleSim[8] and the free modeling, compilation and simulation environment OpenModelica[9].

Even now, tools such as AMESim support Modelica language. Also, the free simulation environment Scicos [39] uses a subset of Modelica for component modeling and Mathworks has launched a similar tool dedicated to physical systems modeling, called Simscape[10].

[5] http://www.lmsfrance.fr/imagine-amesim-suite/
[6] http://www.mathcore.com/products/mathmodelica/
[7] http://www.simulationx.com/
[8] http://www.maplesoft.com/products/maplesim/
[9] https://www.openmodelica.org/
[10] http://www.mathworks.com/products/simscape/

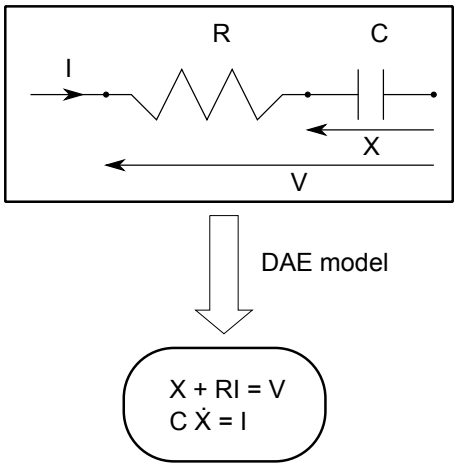

Figure 3.2: Acausal (declarative) models.

The acausal modeling languages make the work simpler for model developer but harder for the designer of the simulation tools. Indeed, they must provide capabilities for symbolic analysis of for example large DAE systems to reduce DAE system to ODE system and make the integration easier.

3.5 Conclusion

This chapter describes the formalization of complex systems (nonlinear hybrid dynamical systems) using differential equations. For example, to design new engine concepts, the combustion phenomenon described by thermodynamic equations uses a set of PDEs for the 3D Computational Fluid Dynamics (CFD) modeling and a set of ODEs for the 0D phenomenological modeling. Besides, this chapter shows also how modeling languages are needed to model complicated intermixed equations (conditions, loop, implicit, etc.). The next chapter exposes the functioning of numerical integrators to solve and simulate these complex models.

Chapter 4

Simulation of physical systems

4.1 Introduction

There are different ways for solving differential equations but they are all based on the same idea, imposed by the limits of the computer: *discretization*. It exists two major families to numerically solve a set of differential equations. The most commonly used integration consists in discretizing the time using time slicing algorithms. The second kind of integration consists in discretizing the state values using another category of algorithms called the Quantized State System (QSS) algorithms. This chapter details each of these techniques.

4.2 Resolution of physical systems through time integration

4.2.1 Purpose of numerical integrators

For numerical (time) integrator, the problem (a set of differential equations) is solved on a grid of parameters. Especially for ODEs, the grid corresponds to the time grid denoted h when it is constant and h_n when it is adaptive. Another notation that can be found in literature is dt. The time grid is called in general "time-step", "integration step" or 'time integration".

Let consider the exact solution $\boldsymbol{X}(t)$ of the real equation (3.1) built by Taylor expansion around the time t_n. It is obtained in time-steps $t_n \to t_{n+1}$ of step-size $h_n = t_{n+1} - t_n$, (n=0,1,...) as follow:

$$\boldsymbol{X}(t_{n+1}) = \sum_{i=0}^{\infty} \frac{h_n^i}{i!} \boldsymbol{X}^{(i)}(t_n), \tag{4.1}$$

with $\boldsymbol{X}$ is of (differentiability) class C^∞, or

$$\boldsymbol{X}(t_{n+1}) = \boldsymbol{X}(t_n) + h_n \left[\boldsymbol{f}(t_n, \boldsymbol{X}(t_n)) + \frac{h_n}{2!} \boldsymbol{f}'(t_n, \boldsymbol{X}(t_n)) + ... + \frac{h_n^{p_n - 1}}{p_n!} \boldsymbol{f}^{(p_n - 1)}(t_n, \boldsymbol{X}(t_n)) + \mathcal{O}(h_n^{p_n}) \right], \tag{4.2}$$

with $\boldsymbol{f}$, the 1^{st} order time-derivative of $\boldsymbol{X}$, is of (differentiability) class C^{p_n}.

The big O notation $\mathcal{O}()$, also called Landau's symbol, describes the asymptotic behavior of a function. For example, a function $\Gamma(h_n)$ is dominated by $\mathcal{O}(h_n^{p_n})$ means that $\exists C > 0$ constant,

such that

$$|\Gamma(h_n)| \leq C h_n^{p_n}. \tag{4.3}$$

The integration consists in approximating as best as possible the exact solution $\boldsymbol{X}(t_{n+1})$ in (4.2) by $\boldsymbol{X}_{n+1}$ using a numerical integrator (or solver).

The estimate $\boldsymbol{X}_{n+1}$ is based on a defined initial conditions $\boldsymbol{X}_0 = \boldsymbol{X}(t_0)$ and on a numerical approximation $\boldsymbol{\Phi}$, also known as increment function.

At the current time t_n, $\boldsymbol{\Phi}$ is (mandatory) function of:

- the current value of the time-step, h_n;
- the current value of 1^{st} order time-derivative, $\boldsymbol{f}_n = \boldsymbol{f}(t_n, \boldsymbol{X}_n)$;
- and the current value of the estimate, $\boldsymbol{X}_n$.

In addition, according to the solver's nature (multi-step, implicit, etc.), defined in section 4.2.3, $\boldsymbol{\Phi}$ could be (optionally) function of:

- past values of 1^{st} order time-derivative, $\boldsymbol{f}_{n-j}$ $(1 \leq j \leq n)$, with $\boldsymbol{f}_i = \boldsymbol{f}(t_i, \boldsymbol{X}_i)$;
- a combination of 1^{st} order time-derivatives at different times between t_n and t_{n+1}, e.g. $\boldsymbol{f}_{n+\alpha} = \boldsymbol{f}(t_{n+\alpha}, \boldsymbol{X}_n + \alpha h \boldsymbol{f}(t_n, \boldsymbol{X}_n))$ such as $0 < \alpha < 1$;
- the future value of 1^{st} order time-derivative, $\boldsymbol{f}_{n+1} = \boldsymbol{f}(t_{n+1}, \boldsymbol{X}_{n+1})$;
- past values of the estimate, $\boldsymbol{X}_{n-j}$ $(1 \leq j \leq n)$;
- and the future value of the estimate, $\boldsymbol{X}_{n+1}$.

For clarity, the description of $\boldsymbol{\Phi}$ will be simplified by mentioning only the term $\boldsymbol{f}$ for all the possible time values described above. Then, the general solution of a numerical solver is defined as

$$\boldsymbol{X}_{n+1} = \boldsymbol{X}_n + h_n \boldsymbol{\Phi}(t_n, \boldsymbol{X}_0, \ldots, \boldsymbol{X}_{n-1}, \boldsymbol{X}_n, \boldsymbol{X}_{n+1}, h_n, \boldsymbol{f}). \tag{4.4}$$

In the following, if the characteristic of the solver is not mentioned, the simple model in (4.5) is considered by default (one-step explicit method).

$$\boldsymbol{X}_{n+1} = \boldsymbol{X}_n + h_n \boldsymbol{\Phi}(t_n, \boldsymbol{X}_n, h_n, \boldsymbol{f}). \tag{4.5}$$

The problem resolution is summarized in finding an integrator accurate, fast and robust. Before going deeper in the subject of the numerical integrators, it is necessary to define the sense of stiff systems. In fact, they are jointly linked to the solvers.

4.2.2 Stiff systems

There exists no unique definition of the term "stiff system". In literature, the definition is explained depending on how the problem is treated.

Definition 1: *"Stiff equations are equations where certain implicit methods, in particular BDF, perform better, usually tremendously better, than explicit ones."* [40, 41]

Definition 2: *"For linear ODEs with time variant Jacobian* $A := (\partial\varphi/\partial x)(x,t)$*, stiffness may often be characterized by the eigenvalues* $\lambda_1, \ldots, \lambda_{n_x}$ *of A: The system is stiff if* $Re\lambda_i \leq 0, (i = 1, \ldots, n_x)$*, and* $\min_i |\lambda_i| \ll \max_i |\lambda_i|$*."* [42]

Definition 3: *"An IVP (Initial Value Problem) is stiff in some interval* $[0,b]$ *if the step size needed to maintain stability of forward Euler method is much smaller than the step size required to represent the solution accuracy."* [43]

Definition 4: *"An ODE system is called stiff if, when solved with any* n^{th} *order accurate integration algorithm and a local error tolerance of* 10^{-n}*, the step size of the algorithm is forced down to below a value indicated by the local error estimate due to constraints imposed on it by the limited size of the numerically stable region."* [11]

The first definition gives an idea on how to choose the best numerical methods when dealing with stiff systems. In other words, explicit methods generally fail or are very slow when they are dealing with stiff problems. Definition 2, also well-known in control engineering, considers that stiffness is only related to differential equations, more exactly on the eigenvalues (or constant time scales) when they are scattered and differ highly in magnitude. Furthermore, definition 3 and 4, add to definition 2, the dependency on accuracy criteria which means that the step-size is forced by the stability conditions rather than by the accuracy conditions.

Next section will detail the different criteria of numerical solvers.

4.2.3 Criteria and parameters of numerical integrators

The numerical integrator aims to solve the system of equations. To select the appropriate one, it is necessary to define the essential criteria to satisfy (e.g. speed, consistency, accuracy, stability, etc.) by acting on the solvers' parameters (e.g. order, implicit, multi-step, etc.).

Speed

For a given model and computing resource, the fastness of a solver depends on the total amount of computations required to resolve the system of equations. The integration speed is the most important criteria to achieve real-time simulation. In this context, the challenge is in particular to find a satisfactory trade-off between the integration speed and precision which usually lead to conflicting constraints.

Order

The order of accuracy of the numerical solution reflects its rate of convergence to the exact solution. A method has an order p (or p^{th} accurate), notated $\mathcal{O}(h^p)$, when it neglects in the Taylor series all terms where the order of derivatives is higher than p (e.g. Euler is 1^{st} order and RK4[1] is 4^{th} order). In the context of order definition, h is either constant for fixed time-step solvers or

$$h = \max_{0 \leq k \leq n} h_k \tag{4.6}$$

for variable step-solvers. For control order algorithms, the order of accuracy is

$$p = \min_{0 \leq p \leq n} p_k. \tag{4.7}$$

[1] Runge Kutta 4

Consistency

The consistency is a property of the discretization that depends on the Local Truncation Error (LTE). The LTE at t_{n+1}, denoted $\boldsymbol{\delta}_{n+1}$, is the error that the approximation $\boldsymbol{\Phi}$ causes during a single time-step h_n, assuming a perfect knowledge of the true solution at the previous time-step

$$\begin{aligned} \boldsymbol{\delta}_{n+1} &= \boldsymbol{X}(t_{n+1}) - \boldsymbol{X}_{n+1}, \\ \boldsymbol{X}_n &= \boldsymbol{X}(t_n). \end{aligned} \tag{4.8}$$

Using the definition of $\boldsymbol{X}_{n+1}$ in (4.5), the LTE in (4.8) can be also written in the following form

$$\boldsymbol{\delta}_{n+1} = \boldsymbol{X}(t_{n+1}) - \left(\boldsymbol{X}(t_n) + h_n \boldsymbol{\Phi}(t_n, \boldsymbol{X}(t_n), h_n, \boldsymbol{f})\right). \tag{4.9}$$

When the method has order p, the local truncation error is proportional to $\mathcal{O}\left(h^{p+1}\right)$, meaning that $\exists C > 0$ constant, such that $\|\boldsymbol{\delta}_{n+1}\| \leq C h^{p+1}$.

The numerical method is consistent when

$$\lim_{h_n \to 0+} \frac{\|\boldsymbol{\delta}_{n+1}\|}{h_n} = 0. \tag{4.10}$$

Numerical stability

The stability represents the amplification/attenuation of the errors during computing (round-off, truncation, method), unconditionally or under conditions (i.e. time-step). It characterizes the propagation of an initial perturbation during an entire numerical integration. [44] presents the regions of stability of different numerical methods. It depends on the eigenvalues of the matrix of amplification $\boldsymbol{A}$ being as $\boldsymbol{X}_{n+1} = \boldsymbol{A}\boldsymbol{X}_n$. The integrator is stable when

$$\|eig(\boldsymbol{A})\| < 1. \tag{4.11}$$

This rule in (4.11) is applied for linear systems because they that can be written in the form of $\boldsymbol{X}_{n+1} = \boldsymbol{A}\boldsymbol{X}_n$. For nonlinear systems, the rule is only valid around equilibrium points t_{eq} where the system linearization is performed (i.e. $\boldsymbol{\Phi}(\boldsymbol{X}(t_{\text{eq}}))$ is transformed to $\boldsymbol{\Phi}_{\text{lin}}.\boldsymbol{X}(t_{\text{eq}}))$.

Explicit/Implicit

For explicit schemes (e.g. explicit Euler, Adams-Bashfort), $\boldsymbol{X}_{n+1}$ is computed directly from past values $\boldsymbol{X}_n, \boldsymbol{X}_{n-1}, ...$ but for implicit schemes (e.g. Cranck Nicholson, Adams-Moulton), the computation of $\boldsymbol{X}_{n+1}$ needs the resolution of an additional equation (often nonlinear).

Compared with explicit schemes, implicit schemes are often less accurate during the initial steps. They are also more complex to implement and require more computations at each time-step, because they need the resolution of a nonlinear system based on Newton iterations at each integration step [45]. Consequently, the cost of each integration step is considerably more expensive than in the case of an explicit algorithm. However, implicit algorithms are far more stable and effective when integrating stiff systems (where the model merges subsystems with very different decay rates and time constants), whereas explicit schemes need to use tiny time-steps to ensure stability, or even totally fail due to numerical instability.

Example

Let consider two solvers, the first is explicit and the second is implicit where

$$\begin{array}{rclcl} \boldsymbol{X}_{\text{exp},n+1} & = & \boldsymbol{X}_n + h_n \boldsymbol{\Phi}(t_n, \boldsymbol{X}_n, h_n, \boldsymbol{f}) & = & \boldsymbol{X}_n + h_n \boldsymbol{\Phi}_{\text{lin}}(t_n, h_n, \boldsymbol{f}) \boldsymbol{X}_n, \\ \boldsymbol{X}_{\text{imp},n+1} & = & \boldsymbol{X}_n + h_n \boldsymbol{\Phi}(t_n, \boldsymbol{X}_{n+1}, h_n, \boldsymbol{f}) & = & \boldsymbol{X}_n + h_n \boldsymbol{\Phi}_{\text{lin}}(t_n, h_n, \boldsymbol{f}) \boldsymbol{X}_{n+1}. \end{array}$$

Their corresponding matrices of amplification are $\boldsymbol{A}_{\text{exp}} = \boldsymbol{I} + h_n \boldsymbol{\Phi}_{\text{lin}}$ and $\boldsymbol{A}_{\text{imp}} = (\boldsymbol{I} - h_n \boldsymbol{\Phi}_{\text{lin}})^{-1}$.

When the system is linear, we can conclude that the explicit scheme has a conditional stability depending on the time-step h_n, however for the implicit scheme the condition $\|eig(\boldsymbol{A})\| < 1$ is always true regardless of h_n.

When the system is nonlinear, the stability for explicit and implicit schemes is under conditions (related to equilibrium points), but it was shown in [43] that the stability of implicit schemes stays valid with larger time-steps than the stability of explicit schemes.

Convergence(accuracy)

The convergence is a property of the numerical solution that depends on the Global Truncation Error (GTE). The GTE at t_{n+1}, denoted $\boldsymbol{\Delta}_{n+1}$, is the accumulation of the local truncation error over all of the $n+1$ time-steps, assuming perfect knowledge of the true solution at the initial time-step

$$\begin{array}{rcl} \boldsymbol{\Delta}_{n+1} & = & \boldsymbol{X}(t_{n+1}) - \boldsymbol{X}_{n+1}, \\ \boldsymbol{X}_0 & = & \boldsymbol{X}(t_0). \end{array} \tag{4.12}$$

The GTE in (4.12) can be also written in the following form

$$\boldsymbol{\Delta}_{n+1} = \boldsymbol{X}(t_{n+1}) - \left(\boldsymbol{X}(t_0) + h_n \sum_{i=0}^{n} \boldsymbol{\Phi}(t_i, \boldsymbol{X}_i, h_n, \boldsymbol{f}) \right). \tag{4.13}$$

The number of time-steps is proportional to $\mathcal{O}(1/h)$ and since LTE is proportional to $\mathcal{O}\left(h^{p+1}\right)$ (the method has order p,), then GTE is proportional to $\mathcal{O}(h^p)$.

A numerical method for the IVP is convergent if for every Lipschitz function $\boldsymbol{\Phi}$ and every $t \in \mathbb{R}_+^*$

$$\lim_{h \to 0+} \max_{1 \leq k \leq n+1} \|\boldsymbol{X}_k - \boldsymbol{X}(t_k)\| = 0. \tag{4.14}$$

The consistency and the stability are necessary and sufficient for the convergence [46]. In other words, a numerical method for the IVP is convergent, if and only if, it is both consistent and stable.

Relationship between LTE and GTE

It is possible to calculate an upper bound on the Global Truncation Error when the Local Truncation Error is already known. Adding and subtracting $\boldsymbol{\delta}_{n+1}$ in (4.13) and using the expression of (4.9), the GTE satisfies then the recurrence form:

$$\boldsymbol{\Delta}_{n+1} = \boldsymbol{\Delta}_n + h_n \left(\boldsymbol{\Phi}(t_n, \boldsymbol{X}(t_n), h_n, \boldsymbol{f}) - \boldsymbol{\Phi}(t_n, \boldsymbol{X}_n, h_n, \boldsymbol{f}) \right) + \boldsymbol{\delta}_{n+1}. \tag{4.15}$$

Next, norms and the triangle inequality are applied to obtain

$$\begin{aligned} \|\boldsymbol{\Delta}_{n+1}\| &\leq \|\boldsymbol{\Delta}_n\| + h_n \|\boldsymbol{\Phi}(t_n, \boldsymbol{X}(t_n), h_n, \boldsymbol{f}) - \boldsymbol{\Phi}(t_n, \boldsymbol{X}_n, h_n, \boldsymbol{f})\| + \|\boldsymbol{\delta}_{n+1}\| \\ &\leq \|\boldsymbol{\Delta}_n\| + h_n \|\boldsymbol{\Phi}(t_n, \boldsymbol{X}(t_n), h_n, \boldsymbol{f}) - \boldsymbol{\Phi}(t_n, \boldsymbol{X}_n, h_n, \boldsymbol{f})\| + \max_{0\leq k\leq n+1} \|\boldsymbol{\delta}_k\|. \end{aligned} \quad (4.16)$$

Assuming now that $\boldsymbol{\Phi}$ is Lipschitz continuous in $\boldsymbol{X}$ i.e.

$$\|\boldsymbol{\Phi}(t_n, \boldsymbol{X}(t_n), h_n, \boldsymbol{f}) - \boldsymbol{\Phi}(t_n, \boldsymbol{X}_n, h_n, \boldsymbol{f})\| \leq L \|\boldsymbol{X}(t_n) - \boldsymbol{X}_n\|,$$

then

$$\|\boldsymbol{\Delta}_{n+1}\| \leq (1 + h_n L) \|\boldsymbol{\Delta}_n\| + \max_{0\leq k\leq n+1} \|\boldsymbol{\delta}_k\|. \quad (4.17)$$

The term $\max_{0\leq k\leq n+1} \|\boldsymbol{\delta}_k\|$ can be also replaced by Ch^{p+1}; h and p are defined in (4.6) and (4.7).

By induction, one can show that $\|\boldsymbol{\Delta}_{n+1}\| \leq \frac{C}{L} h^p \left[(1+hL)^{n+1} - 1\right]$, $n = 0, 1, ...$

Given that $(1+hL)^{n+1} < e^{(n+1)hL} = e^{L(t_{n+1}-t_0)}$, so the GTE is bounded as follow

$$\|\boldsymbol{\Delta}_{n+1}\| \leq \frac{C}{L} h^p \left[e^{L(t_{n+1}-t_0)} - 1\right], \quad (4.18)$$

and finally one can conclude that the bound of the global error is proportional to the maximum local error:

$$\|\boldsymbol{\Delta}_{n+1}\| \leq C_0 \left[e^{L(t_{n+1}-t_0)} - 1\right] \max_{0\leq k\leq n+1} \|\boldsymbol{\delta}_k\|; \quad \text{with} \quad C_0 = \frac{1}{hL}. \quad (4.19)$$

One-step/Multi-step

One-step methods (e.g. Runge Kutta) only use the current values of the state vector $\boldsymbol{X}$ and the derivative $\boldsymbol{f}$, they depend on evaluations of the differential equations at well-chosen locations within the current integration interval. Multi-step methods use current and several past values of $\boldsymbol{X}$ (e.g. Backward Differentiation Formula BDF) and $\boldsymbol{f}$ (e.g. Adams) to achieve a higher order of accuracy. BDF is usually considered as the most effective multi-step method for stiff systems [43]. Newton iteration [11] is used to solve the nonlinear system at each time-step, which represents almost the total cost in solution computation. Adams methods are considered in general as the best known multi-step methods for solving general non-stiff systems, where the nonlinear system is solved by a simple Functional Iteration.

Fixed step/Variable (adaptive) step

In general, the time-step h has to be smaller or sometimes even negligible compared to the system's dynamics in order to achieve stability. An efficiency problem arises when the system's temporal behavior changes over the simulation horizon, for example when fast transients are mixed with slower state evolutions. When using fixed time-step methods, the step-size must be chosen tiny enough to comply with fast dynamics, thus wasting CPU time when integrating the model in its slow behaviors. However, in that case, the number of integration steps can be known (according to the method's order), so that the execution time is predictable in theory. For adaptive methods (e.g. RK45 Fehlberg), the step-size $h_n := t_n \rightarrow t_{n+1}$, is driven by the integration error. A feedback loop adapts the step-size according to the integration error estimate. Iterations and rollbacks are done until a predefined bound on the error is achieved,

thus leading to overheads and unpredictable integration time. However, compared at the same level of accuracy, variable step solvers are faster than fixed time solvers in general.

In other words, the integration step management gives a choice between time driven integration (fixed steps) and error driven integration (variable steps), see details in section 4.2.6.

Error bounds (tolerances)

An error driven solver (e.g. a step-size and/or an order control algorithm) tries, through iterative time-steps and orders selection, that the estimated local error $\boldsymbol{e}$ meets at each time-step a user-defined error tolerance Tol (or desired precision e_d). The error estimate is related to the solver's characteristics as the time-step, the order, the stop condition, etc. (see section 4.2.6).

If each i^{th} solution component $X_{i,n+1}$ $(i = 1, \ldots, n_X)$ does not approach $X_i(t_{n+1})$, the solver tries to reach the relative error tolerance denoted RTol defined as:

$$\left| \frac{X_{i,n+1} - X_i(t_{n+1})}{X_i(t_{n+1})} \right| \leq \mathrm{RTol}_i, \tag{4.20}$$

where RTol_i is the i^{th} component of RTol.

However, when $X_{i,n+1}$ is near to zero, the absolute error tolerance denoted ATol is used because in that case the relative error grows until infinity. ATol is defined as:

$$|X_{i,n+1} - X_i(t_{n+1})| \leq \mathrm{ATol}_i, \tag{4.21}$$

where ATol_i is the i^{th} component of ATol.

Given that $\boldsymbol{e}_{n+1}$ is the vector of estimated local errors $e_{i,n+1}$, computed at t_{n+1}, then the solver stops when either of the two criterion ((4.20) and (4.21)) is fulfilled, in other words, the following condition must be satisfied:

$$e_{i,n+1} \leq \max(\mathrm{RTol}_i \left| X_{i,n+1} \right|, \mathrm{ATol}_i). \tag{4.22}$$

Most step-size control integrators use an error indicator E, such that (4.22) is transformed to

$$\begin{aligned} E_{n+1} &\leq 1, \\ E_{n+1} &= \sqrt{\frac{1}{n_X} \sum_{i=1}^{n_X} \left(\frac{e_{i,n+1}}{\mathrm{RTol}_i \left| X_{i,n+1} \right| + \mathrm{ATol}_i} \right)^2}. \end{aligned} \tag{4.23}$$

In applied problems with $n_X \gg 10$, the individual definition of $2n_X$ error bounds becomes time consuming and it is common practice to use the same error bounds for all the state variables: $\mathrm{RTol}_i = \mathrm{RTol}$, $\mathrm{ATol}_i = \mathrm{ATol}$, $(i = 1, \ldots, n_X)$.

With reference to (4.14), the global error ϵ is proportional to the maximum local error (produced in one single time-step, such that

$$\epsilon_{n+1} = \max_{0 \leq k \leq n+1} \|\boldsymbol{e}_k\| = \max_{0 \leq k \leq n+1} \|\boldsymbol{X}_k - \boldsymbol{X}(t_k)\|. \tag{4.24}$$

Root-finding capability

Root-finding algorithms are needed in numerical solvers when dealing with hybrid ODEs (presence of discontinuities). It is the case when the solution must be stopped at a root of constraint functions, as when a particle trajectory is stopped at the boundary of a geometrical region (e.g. bouncing ball).

Detection of a sign change in (3.6) means that a root exists between two successive integration steps. When a conditional statement (3.6) becomes true (event detection) between two consecutive time-steps (t_n and t_{n+1}), the solver is stopped at t_n and the root-finder algorithm localizes then the time event $t_e \in [t_n, t_{n+1}[$. The accurate location of the root is usually determined by a combination of bisection, interpolation and secant method. The solver is restarted after that with a time-step $h_n = t_e - t_n$ at time t_e and (3.7) and (3.8) are revaluated.

Bisection method [47] is the simplest root-finding algorithm, it is reliable but it has a linear convergence. However, Newton-Raphson and inverse quadratic interpolation methods converge [47] more rapidly (quadratic convergence) but they may not converge when the initial value is far from the root. Besides, the convergence speed of secant method [47] is slower than Newton method, however its behavior is usually fast thanks to its single function evaluation by iteration. Finally, Brent's method [48] is a popular root-finding algorithm that combines bisection, secant and inverse quadratic interpolation methods to take advantage of the fast-converging secant or inverse quadratic interpolation methods when it is possible and the more robust bisection method when it is necessary.

The event location of embedded root-finding algorithms in numerical solvers is usually bounded by a user-defined maximum number of iterations or a user-defined time out.

4.2.4 Basic one-step methods

Euler

Euler (also known as Forward Euler) is the simplest numerical scheme where $\boldsymbol{\Phi}(t_n, \boldsymbol{X}_n) = \boldsymbol{f}(t_n, \boldsymbol{X}_n)$ in (4.5). It is explicit and only a 1^{st} order accurate because $\boldsymbol{\Delta}_{n+1} = \mathcal{O}(h)$, however it is fast thanks to the single call to the derivative $\boldsymbol{f}$. It has a conditional stability: $\|eig(\boldsymbol{A})\| < 1;\ \boldsymbol{A} = \boldsymbol{I} + h\boldsymbol{f}_{\text{lin}}$. The Euler's algorithm is detailed as follow:

Initialize $\boldsymbol{X}_0$, t_0 and h;
while $t_n \leq t_{\text{end}}$ **do**
  Compute $\boldsymbol{X}_{n+1} = \boldsymbol{X}_n + \boldsymbol{f}(t_n, \boldsymbol{X}_n)$, (4.5);
  Increment the time: $t_n = t_n + h$;
end

Algorithm 1: Euler's algorithm.

There are different variants of Euler with different orders of accuracy, where their corresponding equation is summarized in table 4.1:

- *Backward Euler* [43] is a 1^{st} order implicit scheme and it has an unconditional stability since its condition of stability is always satisfied: $\|eig(\boldsymbol{A})\| < 1;\ \boldsymbol{A} = (\boldsymbol{I} - h\boldsymbol{f}_{\text{lin}})^{-1}$;
- *Modified Euler* [43] is a 2^{nd} order explicit scheme thanks to the centered estimator (with 2 calls to the derivative $\boldsymbol{f}$) and presents a conditional stability;

- *Euler-Cauchy* [49] is a 2^{nd} order explicit scheme (with 3 calls to the derivative $\boldsymbol{f}$) and presents a conditional stability.

Table 4.1: Variants of Euler schemes.

Forward Euler	$\boldsymbol{X}_{n+1} = \boldsymbol{X}_n + h\boldsymbol{f}(t_n, \boldsymbol{X}_n)$
Backward Euler	$\boldsymbol{X}_{n+1} = \boldsymbol{X}_n + h\boldsymbol{f}(t_{n+1}, \boldsymbol{X}_{n+1})$
Modified Euler	$\boldsymbol{X}_{n+1} = \boldsymbol{X}_n + h\boldsymbol{f}(t_n + \frac{h}{2}, \boldsymbol{X}_n + \frac{h}{2}\boldsymbol{f}(t_n, \boldsymbol{X}_n))$
Euler-Cauchy	$\boldsymbol{X}_{n+1} = \boldsymbol{X}_n + \frac{h}{2}[\boldsymbol{f}(t_n, \boldsymbol{X}_n) + \boldsymbol{f}(t_n + h, \boldsymbol{X}_n + h\boldsymbol{f}(t_n, \boldsymbol{X}_n))]$

Cranck Nicholson

Cranck Nicholson [50] is a 2^{nd} order implicit scheme based on the trapezoidal rule. It does not explicitly provide $\boldsymbol{X}_{n+1}$, but a more or less complicated equation with $\boldsymbol{X}_{n+1}$. This usually requires using a numerical method for solving nonlinear equations

$$\boldsymbol{X}_{n+1} = \boldsymbol{X}_n + \frac{h}{2}(\boldsymbol{f}(t_n, \boldsymbol{X}_n) + \boldsymbol{f}(t_{n+1}, \boldsymbol{X}_{n+1})).$$

It is a 2^{nd} order implicit scheme and presents a conditional stability.

Runge Kutta

Runge Kutta scheme [43] replaces the evaluation of the derivative at the point $(t_n, \boldsymbol{X}_n)$ by an average of derivatives in the interval h. There are different orders of Runge Kutta. The most "popular" is the Runge Kutta 4 (RK4) in which the calculation of the average of derivatives in the interval h uses four points

$$\begin{cases} \boldsymbol{k}_1 & = \boldsymbol{f}(t_n, \boldsymbol{X}_n) \\ \boldsymbol{k}_2 & = \boldsymbol{f}(t_n + \frac{h}{2}, \boldsymbol{X}_n + \frac{h\boldsymbol{k}_1}{2}) \\ \boldsymbol{k}_3 & = \boldsymbol{f}(t_n + \frac{h}{2}, \boldsymbol{X}_n + \frac{h\boldsymbol{k}_2}{2}) \\ \boldsymbol{k}_4 & = \boldsymbol{f}(t_n + h, \boldsymbol{X}_n + h\boldsymbol{k}_3) \\ \boldsymbol{X}_{n+1} & = \boldsymbol{X}_n + \frac{h}{6}(\boldsymbol{k}_1 + 2\boldsymbol{k}_2 + 2\boldsymbol{k}_3 + \boldsymbol{k}_4). \end{cases}$$

This method is a 4^{th} order explicit scheme and it is one of the most used in relatively simple problems, translated in "soft" curves. It is not suitable, on the other hand, for problems with very large variations of the derivatives.

Its main advantages are the easiness of the implementation and its good stability (even if it has a conditional stability). The issue of accuracy is complex because it is not exclusively related to the order of the method. Theoretically, it is possible to improve it by reducing the integration step, but at the cost of the computation time that becomes quickly prohibitive. Indeed, to perform an integration step, the derivative must be computed 4 times. So when the system presents many variables it becomes very expensive to compute.

4.2.5 Basic multi-step methods

There are three multi-step methods commonly used [43]: Backward Differentiation Formula (BDF) that uses past values of $\boldsymbol{X}$ and Adams-Bashforth and Adams-Moulton that use past

values of $\boldsymbol{f}$. They are all based on the following equation:

$$\boldsymbol{X}_{n+1} + \sum_{j=0}^{s-1} a_j \boldsymbol{X}_{n-j} = \sum_{j=-1}^{s-1} b_j \boldsymbol{f}(t_{n-j}, \boldsymbol{X}_{n-j}),$$

where s is the number of steps of the method, and a_j and b_j are the coefficients that define the method.

Backward Differentiation Formula (BDF)

The BDF methods (or Gear methods) are implicit schemes because $b_{-1} \neq 0$. They are suited to stiff systems. The coefficients are $b_j = 0$, $(j = 0, \ldots, s-1)$, while a_j, $(j = 0, \ldots, s-1)$ are chosen such that the method attains order s.

Adams-Bashforth

The Adams-Bashforth algorithms (predictor-corrector) are explicit methods because $b_{-1} = 0$. The coefficients are $a_j = 0$, $(j = 1, \ldots, s-1)$ and $a_0 = -1$, while b_j, $(j = 0, \ldots, s-1)$ are chosen such that the method attains order s.

Adams-Moulton

The Adams-Moulton algorithms (predictor-corrector) are similar to the Adams-Bashforth, but they are implicit methods because $b_{-1} \neq 0$. The coefficients are $a_j = 0$, $(j = 1, \ldots, s-1)$ and $a_0 = -1$, while b_j, $(j = 0, \ldots, s-1)$ are chosen such that the method attains order $s+1$.

4.2.6 Basic step-size control methods

The issue of time-step solvers is how to choose the adequate step-size h. Either it is too large, leading to errors, instability or divergence, either it is too small, leading to a long computation time. The solver has to find an ideal time-step, that is as large as possible without inducing a big error and that is related to stiffness of the equations.

This ideal time-step needs to have the ability to vary over time (i.e. large in "simple" locations and small in "complex" locations). Indeed, it is clear that if the equation to integrate results in a straight line, one step of integration covering the whole area would be sufficient and would provide an accuracy only limited by round-off errors of the calculator. Conversely, smaller is the curvature radius, smaller the integration step must be chosen to ensure a minimum level of accuracy. That is why variable step solvers are needed rather than fixed step solvers.

Since the exact solution is unknown a priori, it is difficult to estimate the error. The principle of a variable step solver is to compare different evaluations of the integrator, either depending on the time-step, or according to the order of the integrator. When the estimated error e_{n+1} is assessed unacceptable compared to a desired accuracy e_d (or tolerance Tol), the time-step is reduced and a rollback of the integration is performed with this new time-step. This iteration is repeated until satisfying the condition. In the same way, if the produced error is assessed acceptable compared to a desired tolerance, the time-step is increased while satisfying the same condition.

Setting-up such a procedure has a cost in computation time that should be balanced by larger time-steps. A method with adaptive time-step is more complex to implement, but often faster and accurate driven. This, of course, depends on the nature of the physical system (stiffness, etc.).

Adaptive Runge Kutta 4 (RK4)

Adaptive RK4 makes two evaluations of $\boldsymbol{X}$, the first is performed with a time-step h_n and the second is done with two time-steps $h_n/2$. If the difference between the two evaluations e_{n+1} (estimated local error) is equal more or less to the desired precision e_d, then the solution is acceptable and the next time-step is increased, otherwise the time-step is reduced. Here the estimated local error is taken as $e_{n+1} = \max\limits_{1\leq i\leq n_X} |e_{i,n+1}|$.

The new optimal step-size is given by $h_{n+1} = h_n^{\text{new}} = h_n|e_d/e_{n+1}|^\alpha$ where $\alpha \in \mathbb{R}_+^*$. Thus, if $e_{n+1} \leq e_d$ (successful integration step), the step may be increased ($h_n^{\text{new}} > h_n$) for the next step. Conversely, if $e_{n+1} > e_d$, the integration is wrong and it is repeated with a smaller step ($h_n^{\text{new}} < h_n$).

This method is simple but very time consuming. Indeed, one single iteration requires 11 assessments of $\boldsymbol{f}$: 4 with a time-step h_n and $8-1$ with time-step $h_n/2$ (the first evaluation is the same in the two cases). Compared to RK4 with fixed time-step $h_n/2$ (8 assessments), the adaptive RK4 is 37.5% more costly in CPU cycles.

Adaptive Runge Kutta 45: Fehlberg method

This technique is more elegant and faster than adaptive RK4. It is based on the Fehlberg method [43] for RK5 (5^{th} order). Fehlberg studied RK5 which requires 6 calls to the derivative: $\boldsymbol{k}_1 = h_n \boldsymbol{f}(t_n, \boldsymbol{X}_n)$ and $\boldsymbol{k}_i = h_n \boldsymbol{f}\left(t_n + a_i h_n, \boldsymbol{X}_n + \sum\limits_{j=1}^{i-1} b_{ij}\boldsymbol{k}_j\right)$ for $2 \leq i \leq 6$.

Then, $\boldsymbol{X}_{n+1} = \boldsymbol{X}_n + \sum\limits_{i=1}^{6} c_i \boldsymbol{k}_i + \mathcal{O}\left(h_n^6\right)$.

He also found a combination of other coefficients which give a result of 4^{th} order:

$\boldsymbol{X}^*_{n+1} = \boldsymbol{X}^*_n + \sum\limits_{i=1}^{6} c^*_i \boldsymbol{k}_i + \mathcal{O}\left(h_n^5\right)$.

Hence, by computing the same quantities k_1 to k_6, two evaluations of the result can be performed: $\boldsymbol{X}_{n+1}$ at order 5 and $\boldsymbol{X}^*_{n+1}$ at order 4, and the step-size dependent error is at order 5:

$$\max_{1\leq i\leq n_X} |X^*_{i,n+1} - X_{i,n+1}| \approx h_n^5.$$

The new optimal time-step h_n^{new} (or h_{n+1}) is computed as $e_{n+1} = e_d$ (required accuracy).

So $\left(\frac{h_n^{\text{new}}}{h_n}\right)^5 = \frac{e_{n+1}}{e_d} \Rightarrow h_n^{\text{new}} = h_n \left(\frac{e_d}{e_{n+1}}\right)^{1/5}$.

More generally, the error estimate e_{n+1} satisfies (4.22) using a p^{th} order method when the optimal time-step is

$$h_n^{\text{new}} = \alpha_s h_n \left(\frac{e_d}{e_{n+1}}\right)^{1/(p+1)}, \tag{4.25}$$

with $\alpha_s \in [0.8, 0.9]$ as safety factor. This is equivalent to the case where the error indicator E_{n+1} satisfies (4.23) when

$$h_n^{\text{new}} = \alpha_s \frac{h_n}{E_{n+1}^{1/(p+1)}}. \tag{4.26}$$

4.2.7 Step-size and order control methods with root-finder

LSODAR

The most known and used solver with root-finding stopping criteria is LSODAR [45]. It derives from LSODE [51] (Livermore Solver for Ordinary Differential Equations), a basic solver written originally in standard Fortran 77 for the initial value problem for ODE systems.

LSODAR is suitable for both stiff and non-stiff systems. It automatically selects between non-stiff and stiff methods [52]. It uses the non-stiff method initially, and dynamically monitors data in order to decide which method to use. It uses Adams methods (Functional iteration) up to order 12 in the non-stiff case, and BDF methods (Newton iteration) up to order 5 in the stiff case. The maximum order corresponds to the limits of stability of the methods. The switch from Adams to BDF methods is performed when Adams-Moulton method is no longer stable for the problem or cannot efficiently meet the accuracy requirements.

When LSODAR detects a sign change (root detection), it runs its root-finding algorithm. It uses a combination of the secant and bisection methods where the secant method is used by default. Then, after each iteration of the root-finding algorithm, LSODAR evaluates a point on the ODE solution curve by interpolation. The method of interpolation is already defined in the currently used method (Adams-Moulton or BDF).

The solution $\boldsymbol{X}_{n+1}$ is accepted as sufficiently accurate if (4.23) is satisfied. To maintain the desired accuracy while trying to minimize computational work, the solver attempts to change the step-size h_n and/or the method order p_n . To reduce complications associated with p_n and h_n selection, the new order p_n^{new} (or p_{n+1}) is restricted to the values $p_n - 1$, p_n, and $p_n + 1$, where p_n is the current order. For each p_n^{new}, the step-size $h_n^{\text{new}}(p_n)$ is computed under the local error bound condition, while assuming that the highest derivative remains constant. The method order that produces the largest h_n^{new} is used on the next step, along with the corresponding h_n^{new}.

DASRT

DASRT is also a numerical solver with root-finding stopping criteria that derives from DASSL [53] (Differential Algebraic System Solver). DASSL is a basic solver written originally in standard Fortran 77 for the initial value problem for implicit systems of DAEs with index less or equal to one. However, with some modification as described in [54], DASSL (and DASRT) can be used to solve index-two systems.

DASRT is a variable step-size and variable order solver. The derivatives $\dot{\boldsymbol{X}}$ are approximated using BDF methods, with fixed-leading coefficient, of orders 1 through 5. As mentioned earlier, the maximum order corresponds to the limits of stability of the methods. The nonlinear system is solved at each time-step by Newton's method.

For example, if BDF of order 1 is used at time t_n, the original DAEs system in (3.2) is transformed to

$$\boldsymbol{F}(t_n, \boldsymbol{X}_n, \frac{\boldsymbol{X}_n - \boldsymbol{X}_{n-1}}{h_n}) = 0. \tag{4.27}$$

Then, the equation is solved using Newton's iteration:

$$\boldsymbol{X}_n^{m+1} = \boldsymbol{X}_n^m - \left(\frac{\partial \boldsymbol{F}}{\partial \dot{\boldsymbol{X}}} + \frac{1}{h_n}\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{X}}\right)^{-1} \boldsymbol{F}(t_n, \boldsymbol{X}_n^m, \frac{\boldsymbol{X}_n^m - \boldsymbol{X}_{n-1}}{h_n}), \tag{4.28}$$

where m is the iteration index. The convergence is faster when the initial guess $\boldsymbol{X}_n^0$ is accurate. This initial guess is obtained by interpolation using past values ($\boldsymbol{X}_{n-1}$, $\boldsymbol{X}_{n-1}$, etc.).

DASRT is useful for DAEs problems with discontinuities. The solver stops at root of user-prescribed function (as for LSODAR) and uses the root-finder to locate the discontinuity. Then, it restarts with a new function.

4.3 Resolution of physical systems through state quantization

4.3.1 Introduction

Initially, Zeigler [55] introduced the Discrete Event System Specification (DEVS) formalism in order to enable the discretization of states. Then, Kofman improved this approach and developed a first-order non-stiff QSS algorithm [56] in 2001.

Whereas classic numerical solvers, that use time discretization, convert ODE systems to equivalent difference equation systems, QSS solvers convert the continuous-time model to an equivalent discrete-event model.

Originally, QSS algorithms were implemented under DEVS simulation engines such as PowerDEVS. However recently, in order to overcome the problem of the large overhead introduced by some features of DEVS engines, a family of stand-alone QSS solvers was developed. They are faster and can be directly implemented in a chosen environment such as OpenModelica[2].

4.3.2 DEVS formalism

DEVS [55] is a formalism which allows to represent and simulate any system with a finite number of changes in a finite interval of time. In that way, systems modeled by Petri Nets, State Charts, Event Graphs, and even Difference Equations can be seen as particular cases of DEVS models.

A DEVS model treats an input event trajectory and produces an output event trajectory according to this input and its own initial conditions. An atomic DEVS model is defined as follow:

$$M = (\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{S}, \delta_{\text{int}}, \delta_{\text{ext}}, \lambda, ta), \tag{4.29}$$

[2] `https://www.openmodelica.org/`

where $\boldsymbol{X}$ is the input event vector, $\boldsymbol{Y}$ is the output event vector, $\boldsymbol{S}$ is the state vector and δ_{int}, δ_{ext}, λ and ta are functions that define the system dynamics:

- $ta(s)$ $(ta(s) : \boldsymbol{S} \to \mathbb{R}_+)$ $with$ $s \in \boldsymbol{S}$ is the time advance function. It specifies how long the system stays in a given state s when there is no input events. $ta(s_1)$ is performed when the state takes up the value s_1 at time t_1;
- δ_{int} $(\delta_{int} : \boldsymbol{S} \to \boldsymbol{S})$ is the internal transition function. $\delta_{\text{int}}(s_1)$ is carried out at time $t_2 = t_1 + ta(s_1)$ and changes the state s_1 to a new one s_2;
- λ $(\lambda : \boldsymbol{S} \to \boldsymbol{Y})$ is called the output function. $\lambda(s_1)$ is performed at a state transition (from s_1 to s_2) and produces an output event y_1

  $\Rightarrow$ The functions ta, δ_{int}, and λ define the autonomous behavior of a DEVS model;
- δ_{ext} $(\delta_{\text{ext}} : \boldsymbol{S} \times \mathbb{R}_+ \times \boldsymbol{X} \to \boldsymbol{S})$ is the external transition function. $\delta_{\text{ext}}(s_2, e, x_1)$ is carried out when an input event x_1 comes at time $t_2 + e$ before the time $t_2 + ta(s_2)$ $(ta(s_2) > e)$ and it changes the state s_2 to a new state s_3. However, it does not produce an output event.

Note here that the nomenclatures $\boldsymbol{X}$ and $\boldsymbol{Y}$ will only be valid for this section 4.3 on state quantization to keep the same definitions in [55]. They are not related to the state and output vectors. Figure 4.1 illustrates the behavior of a DEVS model.

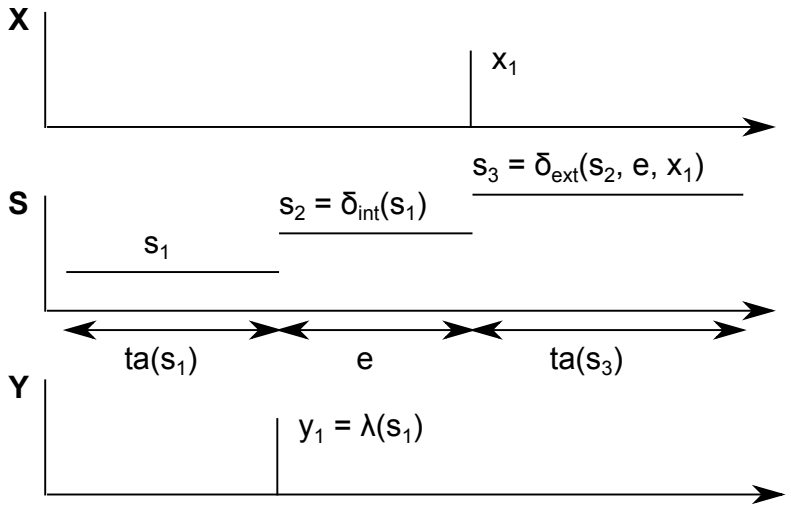

Figure 4.1: DEVS trajectories.

Atomic DEVS models can be coupled (as for block diagrams) in order to form a new DEVS model that can itself be interpreted as an atomic DEVS and can be coupled too with other atomic or coupled models. This approach is usually employed for complex systems to be represented by a hierarchical DEVS.

4.3.3 Principle of Quantized State Systems

Given the ODE system (3.4a), it will be rewritten in the form $\dot{\boldsymbol{x}} = \boldsymbol{f}(t, \boldsymbol{x}, \boldsymbol{u})$ to avoid ambiguity with (4.29). Then, it is approximated as follow:

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(t, \boldsymbol{q}, \boldsymbol{v}), \tag{4.30}$$

where $\boldsymbol{q}$ and $\boldsymbol{v}$ are respectively the quantized variables of $\boldsymbol{x}$ and $\boldsymbol{u}$ and their values correspond to the next lower quantized value of $\boldsymbol{x}$ and $\boldsymbol{u}$ depending on parameters Δq_i (or Δx_i),$(i = 1, \ldots, n_x)$, called quantum. For example, if $\Delta q_i = 1$ and $x_i(0) = 10.5$ then $q_i(0) = 10$.

Each state x_i may reach the next threshold $x_i + \Delta q_i$ at a time different from the others depending on its gradient. This leads to a natural asynchronous behavior (the time-steps are not common to all the variables). Besides, the information concerning the state that cross its next threshold must be transferred only to integrators that need it (integrators that process the state equations which depend on the concerning variable).

As long as none of the states x_i $(i = 1, \ldots, n)$ crosses its next threshold $x_i + \Delta q_i$, all state derivatives $\dot{x_i}$ stay constant and all the state variables x_i are linear functions of time.

- If $\dot{x_i} > 0$ then $q_i = q_i + \Delta q_i$ and x_i increases linearly to $x_{i,k+1}$;
- If $\dot{x_i} < 0$ then $q_i = q_i - \Delta q_i$ and x_i decreases linearly to $x_{i,k}$;
- If $\dot{x_i} = 0$ then q_i and x_i remains constant.

QSS methods have the potential of being efficient with systems which have many discontinuities. In fact, for time slicing method, in order to handle discontinuities, an evaluation of a zero-crossing function (root-finding solver) is needed at each time-step. The solver detects a discontinuity when the sign of this function changes. Then an iterative process (previously described in section 4.2.3) is launched to detect the exact time of that event. Whereas for state discretization method (QSS), the discontinuities are handled naturally. In fact, it does not need to locate discontinuities by invoking an iteration algorithm (root solver) which provokes overheads neither by interpolating between sampling instants (use of a dense output feature).

QSS solver principle is to convert a continuous-time model to an equivalent discrete-event model, so it detects when a solution passes through a given threshold which is the case of discontinuities. QSS solvers could be then interesting for real-time simulation with heavily discontinuous systems.

QSS solvers satisfy practical stability, convergence and error bound properties (see [57] and chapter 12 in [11]). In fact, they preserve numerical stability because the quantization process is seen as a bounded perturbation on the original ODE. They also offer a global error bound, which means that for linear time-invariant analytically stable systems, the difference between the numerical solution and the analytical solution will be bounded.

Besides, QSS solvers belong to explicit, asynchronous, variable time-step and fixed order solvers category. Unfortunately, they do not always work. They lead to frequently switch discrete-event models, that switch boundlessly within a finite time period.

Hysteretic quantization function

In order to avoid the generation of illegitimate discrete-event models, an hysteresis is introduced in [57] as follow. It prevents the infinite switching frequency behavior.

Let $\boldsymbol{D} = \{d_0, d_1, \ldots, d_n\}$ be a set of real numbers where $d_{k-1} < d_k$ $(1 \leq k \leq r)$. Let $x \in \Omega$ be a continuous trajectory where $x : \mathbb{R} \to \mathbb{R}$. Let $b : \Omega \times t_0 \to \Omega$ be a mapping and assume that

$q = b(x, t_0)$ satisfies:

$$q(t) = \begin{cases} d_m & \text{if } t = t_0 \\ d_{j+1} & \text{if } x(t) = d_{j+1} \quad \text{and} \quad q(t^-) = d_j \quad \text{and} \quad j < r \\ d_{j-1} & \text{if } x(t) = d_j - \varepsilon \quad \text{and} \quad q(t^-) = d_j \quad \text{and} \quad j > 0 \\ q(t^-) & \text{otherwise} \end{cases}$$

with

$$m = \begin{cases} 0 & \text{if } x(t_0) < d_0 \\ r & \text{if } x(t_0) \geq d_r \\ k & \text{if } d_k \leq x(t_0) < d_{k+1} \end{cases}$$

Here b is the quantization function with hysteresis (see figure 4.2) and d_0 and d_r are the lower and upper saturation values.

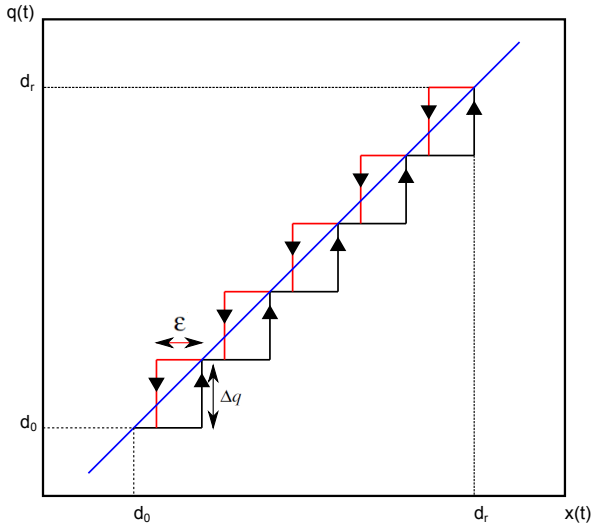

Figure 4.2: Quantization function with hysteresis.

A fundamental property is given by the following inequality:

$$d_0 \leq x(t) \leq d_r \Rightarrow |q(t) - x(t)| \leq \max_{1 \leq i \leq r} (d_i - d_{i-1}, \varepsilon).$$

If the quantization intervals are uniform

$$|\Delta x| \leq \max(\Delta q, \varepsilon). \tag{4.31}$$

The smaller the hysteresis width ε is chosen, the higher may be the oscillation frequency. However if the hysteresis width ε becomes larger than the quantum Δq then the error increases. Usually, the quantum and the hysteresis width are chosen equal to each other because it is the best choice that take into account the trade-off between error and computational costs. In fact, as it is shown in (4.31) the final error in the simulation is bounded by a value proportional to the larger of the hysteresis width and the quantum size. Thus, if the hysteresis width is taken equal to the quantum size, the switching will be reduced without increasing the error bound, $|\Delta x| \leq \Delta q$.

Henceforth, for each $i \in [1 \ldots n_x]$, q_i and x_i are related by the following hysteretic quantization function:

$$q_i(t) = \begin{cases} x_i(t) & if \quad \left| q_i(t^-) - x_i(t) \right| \geq \Delta q_i, \\ q_i(t^-) & otherwise. \end{cases}$$

Logarithmic quantization

Usually, the discrete values q_i are equidistant. However, the use of uniform quantization implicitly controls the absolute error and it would be better to have control over the relative error. This issue might be resolved with the use of logarithmic quantization [58].

4.3.4 QSS solvers

QSS1

QSS1 is a 1^{st} order accurate QSS algorithm [56]. Between two events, the discretized states $q_i(t)$ are constant, which lead to constant state derivatives $\dot{x_i}$ $(i = 1, \ldots, n_x)$. So, the state variables x_i are linear functions of time.

QSS1 is limited because keeping the simulation error small requires a large number of steps and the number of events grows inversely proportional to the discretization Δq.

Higher order QSS

QSS2 shares the same main properties and advantages of QSS1, but it is a 2^{nd} order accurate QSS algorithm [59]. Between two events, the discretized states $q_i(t)$ are linear functions of time, which lead to nonlinear state derivatives $\dot{x_i}$. By linearizing them around the current state, the state variables x_i become parabolic functions of time. The number of events grows inversely proportional to the square root of the discretization Δq.

QSS3 is a 3^{rd} order accurate QSS algorithm [60]. Between two events, the discretized states $q_i(t)$ are parabolic functions of time, which lead to nonlinear state derivatives $\dot{x_i}$. By linearizing them around the current state, the state variables x_i become cubic functions of time. Following the same methodology, the number of events grows inversely proportional to the cubic root of the discretization Δq.

4.3.5 QSS solvers for stiff systems

Advantage of state discretization over time slicing

For the time slicing method, the update of state variables is done in a synchronous way, which means that the time-step is chosen in terms of the fastest changing variable. In a stiff system with widely spread eigenvalues, where slow and fast variables coexist, the slowly changing state variables have to be updated much more often than necessary. This configuration increases significantly the simulation computation time.

However, for the state discretization method, the update of state variables can be done in asynchronous way, which means that each state variable can be updated at its own rhythm and when an event triggers its evaluation. Besides, in a sparse system, when a state variable x_i changes, only the time-derivatives f_j that depend on this state variable x_i have to be reevaluated. This leads to an additional important reduction of the computational costs.

BQSS

BQSS (Backward QSS) is a 1^{st} order solver intended to stiff systems introduced in [61] that shares the main properties of QSS (practical stability, global error bound, etc.). It is considered as an explicit method because there is no iterations as Newton iteration for implicit time slicing algorithm.

The idea is that each quantized variable q_i has always a future value of the corresponding state x_i, so each q_i has only two possible future values, one from below of x_i ($q_i - \Delta q_i$) and the other from above of x_i ($q_i + \Delta q_i$).

This method is limited in accuracy because of its order and there is no way to extend it to higher order. Besides, in some nonlinear systems, BQSS finds non-existing equilibrium points due to the introduction of an extra perturbation term that enhance the error bound.

LIQSS

LIQSS (Linearly Implicit QSS) [62] is a 1^{st} order accurate solver that combines the idea of BQSS and linearly implicit integration. It follows the principle of BQSS, but avoids its mentioned drawbacks (perturbation term, spurious equilibrium point) by using a linearly implicit idea to locate the state values where some time-derivatives pass through zero.

Higher order LIQSS

LIQSS2 [62] and LIQSS3 are respectively 2^{nd} and 3^{rd} order solvers that aim to improve accuracy. They are the combination of QSS2 (QSS3) and LIQSS principles. The number of steps grows with the square (cubic) root of the accuracy.

4.3.6 QSS solvers for marginally stable systems

CQSS

CQSS [63] (Centered QSS) is a 1^{st} order geometric solver that deals with marginally stable systems [64]. Marginally stable systems have their dominant eigenvalues widely spread along the negative axis of the complex plane.

Forward and Backward Euler can be combined to form an F-stable integration method (the Trapezoidal Rule). Similarly, QSS and BQSS can be combined by taking, for each quantized variable, the mean value of the corresponding QSS and BQSS quantized variables, namely, $q_i = \frac{1}{2}(q_{i,\text{QSS}} + q_{i,\text{BQSS}})$.

Just like the previous QSS algorithms, the method does not call for Newton iterations. It is also well-suited for real-time simulation of marginally stable systems with a modest accuracy requirements.

CQSS is limited in terms of accuracy, however it is mentioned in [65] that it is hard to construct higher-order CQSS methods.

4.3.7 QSS tools

QSS solvers under DEVS engines

QSS solvers are integrated in the DEVS engine where each component of (4.30) is seen as two subsystems:

- a static one that includes a static function (F_i):

$$\dot{x_i}(t) = f_i(q_1, ..., q_{n_x}, v_1, ..., v_{n_u}) \tag{4.32}$$

- a dynamical one that includes a hysteretic quantized integrator (HQI) Q_i:

$$q_i(t) = Q_i(x_i(.)) = Q_i(\int \dot{x_i}(\tau)d\tau) \tag{4.33}$$

where Q_i is function of the trajectory $x_i(.)$ and not of the instantaneous $x_i(t)$.

As $v_i(t)$, $q_i(t)$, and $\dot{x_i}(t)$ are piece-wise constant, the input and output of both subsystems (4.32) and (4.33) can be represented by sequences of input and output events. Then, these subsystems can be equivalent to DEVS models, which means that solving (4.30) by the QSS can be exactly simulated by a DEVS model consisting in the coupling of n_x quantized integrators, n_x static functions, and n_u signal sources as figure 4.3 shows.

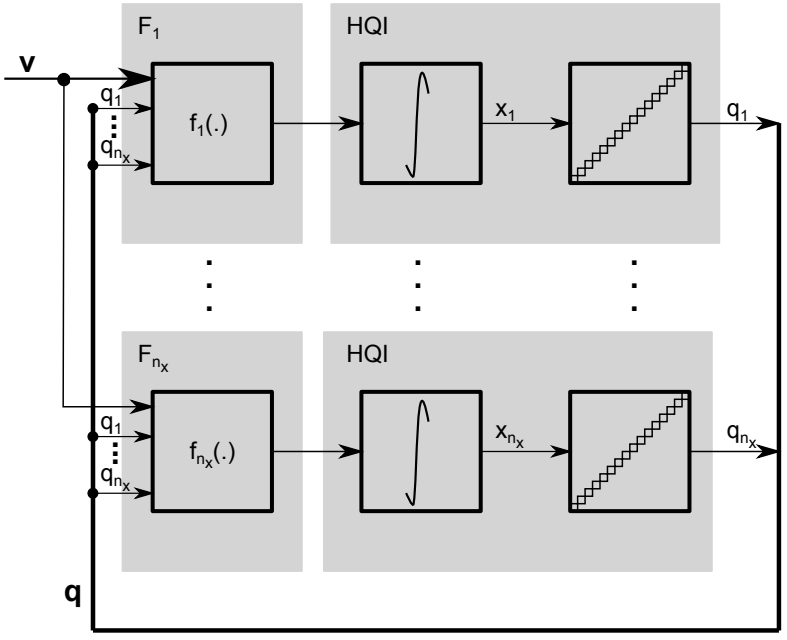

Figure 4.3: Coupled DEVS models of a QSS approximation.

Stand-Alone QSS solvers

The stand-alone QSS solver implements all QSS algorithms without the use of a DEVS engine. It includes two parts:

- Simulation engine that integrates the equation $\dot{\boldsymbol{x}} = \boldsymbol{f}(t, \boldsymbol{q})$ assuming that $\boldsymbol{q}(t)$ is given;
- Solvers that compute $\boldsymbol{q}(t)$, knowing $\boldsymbol{x}(t)$.

In the simulation engine, the model must be given in order to allow that each component of $\boldsymbol{f}(t, \boldsymbol{q})$ can be evaluated individually. In the same way, each zero-crossing condition must be given separately with its event handler. This is necessary to exploit optimally the asynchronous behavior of the QSS. Besides, information about the dependencies between variables and equations (incidence matrices) must be given too. Given all these information a translator implemented in the tool generates the model described in a C language interface needed by the stand-alone QSS solvers for simulation.

4.4 Conclusion

This chapter studies different families of numerical integration methods, their convergence, accuracy and stability. Numerical solvers based on time integration are the focus of the thesis work and present the first step of our investigation. QSS solvers development is recent compared to the traditional time integrators, however the study of this kind of solver is interesting especially for systems with many discontinuities, even if its application is relatively little used. The next chapter presents different kind of parallelization for both kind of solvers.

Chapter 5

Parallelization approaches for ODEs and hybrid ODEs resolution

5.1 Introduction

When complex hybrid ODEs have to be solved, simulation duration becomes of primary concern. To allow a speed-up of this simulation and to avoid model simplification, parallelization of the simulation resolution is of interest.

Parallel computing is employed for solving large problems that could be partitioned into many independent small parts in order to be solved by multiple processing elements in a simultaneous way.

5.2 Levels of parallelism

There are different types of parallelism: bit-level, instruction level, data and task or thread parallelism.

5.2.1 Bit-level parallelism

The bit-level parallelism is directly related to the processor word size. In fact, when variables' sizes are greater than the length of the word, the processor have to execute an operation in at least two instructions. So increasing the word size will reduce the number of instructions.

5.2.2 Instruction-level parallelism

The instruction-level parallelism is directly related to the processor pipeline size. In fact, the number of stages in the pipeline represents the potential parallelism for the instructions. In order to have an efficient parallelism, there have to be many independent instructions that could be re-ordered and grouped together into a pipeline. Recent processor have super-scalar capabilities allowing them to be able to execute several instructions in parallel, using hardware algorithms like Tomasulo algorithm.

5.2.3 Data parallelism

The data parallelism is directly related to program loops. It is possible only when there are no dependencies in the iteration loops. The iterations could be then divided into the number of available processors and executed in parallel without disturbing the data.

5.2.4 Task parallelism

The task parallelism consists in distributing threads (or process) across multiple core or processors. It allows different calculations on the same or different sets of data, unlike the data parallelism which only allows the same calculation on them.

In this thesis we are interested by this kind of parallelization level.

5.3 Parallelization approaches using numerical time integration

Burrage proposed a classification into three categories of the methods for the parallel solution of ODEs [66], that is still valid for hybrid ODEs or DAEs.

5.3.1 Parallelization across the method

The main approach to obtain a parallel numerical scheme for ODE systems consists of parallelizing "across the method". The main idea is to exploit concurrent function evaluations (like state derivatives) within an integration step. Burrage provides an excellent review of these methods in [67].

Explicit multi-stage Runge-Kutta methods are sequential. However, in some cases, two or more stages of Diagonally Implicit Runge-Kutta (DIRK) 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 [68]. The conclusion is that the parallelization potential is limited.

In the sixties, Miranker and Liniger [69] have proposed a framework that allows devising parallel predictor/corrector methods. Later generalizations are reviewed in surveys [70] and [67]. Like the previous approaches, these methods offer a limited parallelization potential. One drawback of these methods 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 [71] or parallelizing operations on vectors for ODEs resolution by separating them into modules (see PVODE solver [72] implemented using MPI (Message-Passing Interface) technology).

Domain specific approaches were also studied. In [73], the parallel execution of multi-body simulations on multi-core or shared memory multi-processors was studied. Using OpenMP[1], the proposed approach relies on the parallelization of matrix/vector operations. It was implemented on the MBSim tool and evaluated on several simple and complex case-studies.

[1] `http://www.openmp.org/`

5.3.2 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 PDEs using the multi-grid approach [74]. Among the techniques introduced in this area, the Parareal scheme proposed in [75] and PITA algorithm described in [76], where both of them were derived from the multiple shooting method [77]. 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. In [78], adaptivity in the tolerance of the time slice integrator and the number of sub-domains was studied and some improvements has been shown towards stiff ODEs. However, for very stiff problems, difficulties still appears. So another parallel solver has been proposed for stiff ODEs based on Richardson Extrapolation.

5.3.3 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 include the relaxation algorithm, the transmission-line modeling method and the modular time-integration.

Waveform Relaxation

Waveform Relaxation (WR) method was introduced in [79]. It is an iterative process originating from Picard theorem, which makes possible to solve simultaneously in parallel coupled subsystems over successive time windows. Each subsystem is characterized by its waveform (i.e. its solution over a determined time interval). The purpose is to find the waveform of a subsystem, considering all the waveforms of the other subsystems constant during one iteration. For practical results [80], a sequential Gauss Seidel and a parallel Gauss Jacobi WR codes have been developed. The second implementation is considered in the sequel.

Given a system partitioned into two subsystems with x_1 and x_2 the two waveforms:

$$\frac{dx_1}{dt} = f(t, x_1, x_2), \quad x_{10} = x_1(t = 0) \tag{5.1}$$

$$\frac{dx_2}{dt} = f(t, x_1, x_2), \quad x_{20} = x_2(t = 0) \tag{5.2}$$

The initialization is done by freezing, during all simulation time, x_{20} for the first WR and x_{10} for the second WR. Then, the first iteration is done by integrating simultaneously (5.1) and (5.2) and saving in memory the trajectories $(x_1^1, x_1^2, \ldots, x_1^n)$ and $(x_2^1, x_2^2, \ldots, x_2^n)$ at the communication time-step. Each subsystem can be integrated with a variable-step or a fixed-step solver but all must use the same communication time-step.

If the tests of convergence are satisfied for iteration it:

$$|x^i_{1,it} - x^i_{1,it-1}| <= \epsilon \quad \text{and} \quad |x^i_{2,it} - x^i_{2,it-1}| <= \epsilon; \quad \text{for} \quad i = [1 \dots n], \tag{5.3}$$

then the integration is successful. Otherwise another integration is restarted, using the already computed state trajectories updated from the other subsystems, until convergence (see figure 5.1).

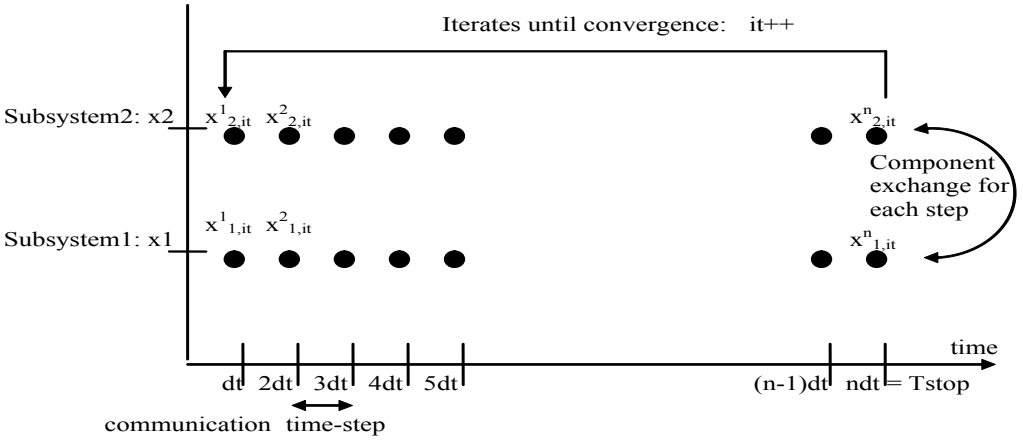

Figure 5.1: Waveform relaxation technique.

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 [81], three types of coupling methods were compared to show how they affect the convergence of the solution. Besides, it is shown in [80] that longer is the simulation time until communication between the WR, slower is the convergence. In this context, several parameters were studied in order to improve the number of iterations:

- Control of the solver tolerance depending on the WR iteration. In fact, for the first WR iterations, the results are not accurate because of the lack of data coming from other sub-systems. The idea is then to relax the solver tolerance just for the first iterations, then progressively tighten it;
- Initialization with infinitesimal disturbance of equilibrium position. Indeed, with initialization close to the solution, few iterations will be need to converge;
- Using *time windows* for the WR. In fact, integrating until a specified time window instead of the end of simulation eases the convergence and decreases the number of iterations. However, it is difficult to find the optimal window size that takes into account the (variable) system's dynamic behaviors;
- Computing on-line adaptive *time windows* whose sizes depend on the development of the waveforms (based on previously computed solution), e.g. the windows can be rescaled using the ratio between convergence tolerance of two successive iterations [80].

Transmission Line Modeling TLM

The Transmission Line Modeling TLM [81] 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. The principle is to introduce impedances between components that cause physically motivated time delays.

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). Other advantages of this technique are the ability to increase the efficiency and the accuracy of the solution, in fact the discrete model is derived with TLM directly from the physical system (elimination of the discretization error).

The Hopsan [82] simulation tool, introduced in the late 70s, is used primarily for hydro-mechanical simulation. It allows multi-domain system modeling and coupled simulation. It integrates the TLM method allowing to perform multi-threaded and multi-core simulations.

The integration of TLM method in Modelica was studied in [83]. Besides, an automatic algorithm for partitioning models, based on TLM, has been developed in [84]. Moreover, in [85], the TLM method is used for exploiting multi-core hardware in real-time and embedded systems.

Modular time-integration or co-simulation

We are interested in this thesis by this kind of parallelization method. A modular time integration method, also called co-simulation, sees the system to be integrated as a connection of several subsystems. Its proceeds in macro steps. 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 [86]. The xMOD tool [87] supports this method. In xMOD, each subsystem (that can be an FMU or a model from an authoring tool like Simulink), is assigned to a thread, with an associated communication step-size and solver. This method was benchmarked by Faure et al. [4] in the context of Hardware In the Loop. Several alternative methods were proposed to perform real-time simulation of complex physical models. However, the study was focused only on fixed-step solver without treating the case of variable-step solver, then we extended it in [88] to examine the case of variable time-step solvers.

In [89], 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 [90], and an approach for the decomposition into weakly coupled components was proposed in [91].

5.4 Parallelization approaches using numerical state quantization

5.4.1 Introduction

As it was shown before, QSS is asynchronous, i.e. the communication between processes is staggered over time. This makes the distribution less complex to perform. Besides, when a communication is required (when the next threshold is crossed), it is performed selectively between only the involved processes which may decrease even more the communication frequency. Finally, the data exchanged is quite small due to the fact that only a small information about

the increase or the decrease by one of the actual level is needed to communicate, which may lead to faster communication.

5.4.2 Parallel Discrete Event Simulation (PDES)

Parallel Discrete Event Simulation is a technique used for multi-core simulation of large DEVS models. The subsystems or the physical processes, that form the original complex system, are simulated concurrently on different Logical Processors (LPs).

Thanks to PDES, the computational cost is reduced compared to a sequential execution. However, in the same time it introduces the need of synchronization between processes. In fact, when a process depends on another process's results, synchronization between them is required in order to simulate correctly its own subsystem. If the synchronization is not done correctly, this will lead to incorrect results. Inaccurate synchronization occurs when the causality constraint is not respected, i.e. receiving out of order events (time-stamps lower than the actual simulation time).

5.4.3 Basic approaches of PDES synchronization

There are many algorithms for PDES synchronization that are derived from the following basic approaches:

- **Conservative synchronization**: Introduced in [92] where all the processes communicate together without a shared memory but through messages managed by a centralized coordinator. The correct order of all messages is achieved by enforcing all LPs to wait until it is safe to produce the next event, i.e. when there is no risk to receive out of order messages. This waiting provokes a reduction of the parallel computing benefits;
- **Optimistic synchronization (Time Warp)**: Introduced in [93] where the causality constraint is relaxed, which means that the LPs are authorized to advance their simulation time as fast as they can. when an out of order event is detected, the LP rolls back to a previously saved state corresponding to a safe state and un-sends all events which were sent out during the rollback period. This leads to expensive computations. Besides saving the states of each LP for possible rollbacks leads to large memory requirements;
- **NOTIME**: Introduced in [94] where there is no synchronization between processes, so the parallelism is exploited optimally. This introduces errors in the simulation. This technique could be performed when simulation speed is more important than simulation accuracy.

Several PDES techniques were based on one of these mentioned basic approaches, but in [95], it is shown that, for DEVS models resulting of the application of QSS to a large system, each of these basic approaches is not well-suited for this kind of problem. To overcome the difficulties they introduced a non-strict synchronization described below.

5.4.4 Scaled Real-Time Synchronization (SRTS)

Scaled Real-Time Synchronization was introduced in [95], where the basic idea to optimize synchronization and to avoid overhead due to inter-process synchronization, is to opt for syn-

chronizing each process's simulation time with a scaled version of the physical time (wall-clock time) instead of synchronizing the simulation time between all processes.

As all processes are synchronized with the physical time, they are indirectly synchronized against each other. Therefore, the only communication between processes occurs when events are emitted from one process to another.

The synchronization with the wall-clock is performed as follow. When an event is attempted to occurs after τ units of simulation time, the processes have to wait until the wall-clock advances τ/r units of physical time (see figure 5.2). The parameter r is the real-time scaling factor.

The real-time scaling factor must be well-chosen. In fact, if it is too big, this will lead to overrun situation but in the same time the simulation will be fast and if it is too small, the overruns will be minimized at the cost of the simulation speed.

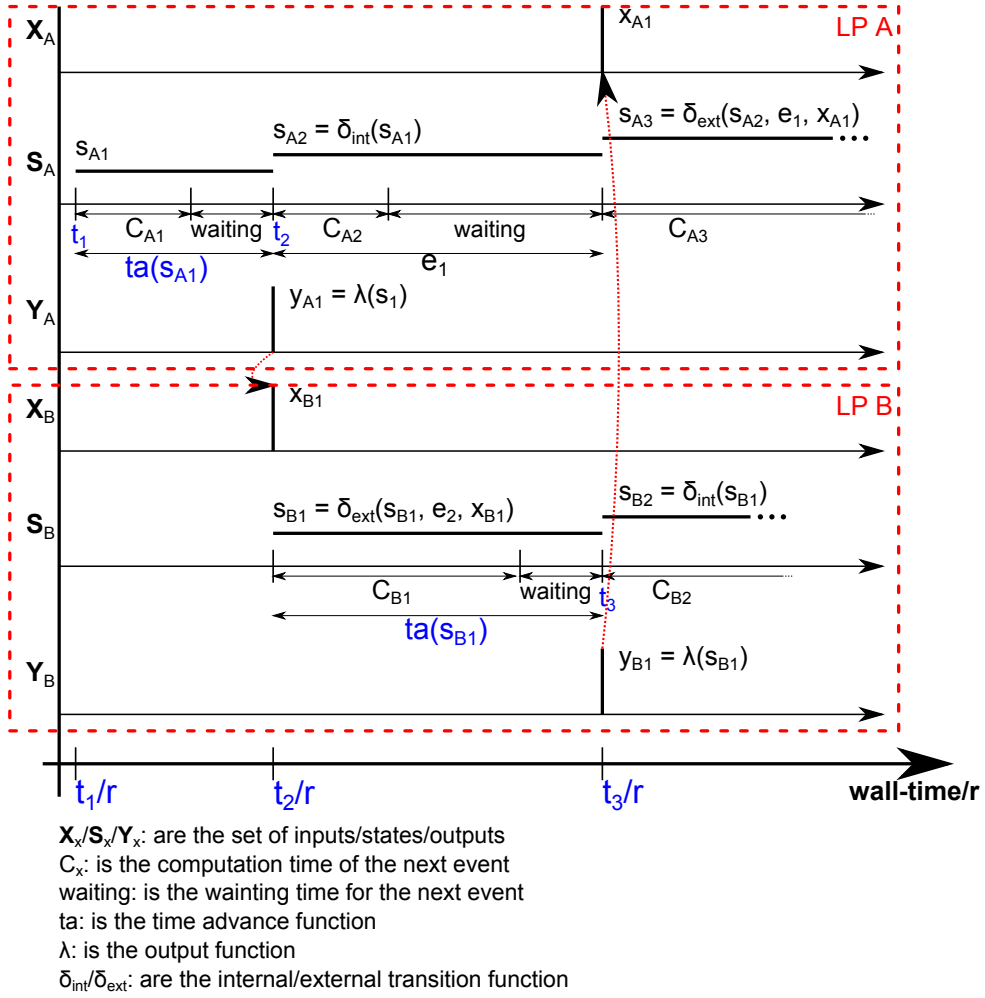

Figure 5.2: Time diagram with SRTS.

5.4.5 Adaptive Real-Time Scaling (ARTS)

Adaptive Real-Time Scaling [95] is an Adaptive-SRTS (ARTS) where the scaling factor r is adjusted dynamically each sampling time (thanks to periodic checkpoints) depending on the system workload. The computation of r is done by a thread coordinator while the others threads wait in a synchronizing barrier.

Adjusting the real-time scaling factor, by either slowing down or increasing the simulation speed, allows the ARTS to drive the minimum waiting ratio to the desired value. The adjustment is performed through tuning parameters:

- The sampling period should be larger than the time spent for re-synchronization, but small enough to be able to react in time to workload change;
- The desired waiting ratio should be chosen small enough to minimize the waiting time but not very close to 0 to avoid overruns, so numerical errors;
- The discrete eigenvalue is a parameter that defines the speed of the adaptation of the real-time scaling factor (close de 0 means fast and close to 1 means slow and smooth).

5.5 Conclusion

This chapter describes the different levels of parallelism to position, at the end, our work at the thread-level parallelism. Then, different approaches of parallelization are exposed, for both numerical time integration methods and numerical state quantization methods, already described in chapter 4. In fact, our major contributions in this thesis will investigate the modular time integration (co-simulation), while keeping in mind the possibilities afforded by the other methods.

Chapter 6

Statement of work

This chapter aims to articulate the proposed state of the art, designed around the thesis work, with the contributions in the context of IFP Energies nouvelles.

Part II, about "context and problem position", gives a state of the art that is related to the thematic covered by the thesis work, i.e. the validation of complex systems through simulation. It introduced the different encountered problematics, the proposed existing approaches and solutions to cope with them, and their current limits and bounds.

This state of the art is non-exhaustive regarding the different involved fields, i.e. modeling, simulation, real-time and parallelism, since it would be too large and diverse to cope with this thesis manuscript. However, the state of the art is introduced in a way that guide and orientate the reader to understand the context of the work and also how and why we position the problem.

It is important to be aware that we are dealing with complex systems whose modeling implies a systemic and holistic approach. A complex system is seen as system of systems, meaning that it is a group of interacting and interdependent components.

The systemic simulation implies the use of 0D phenomenological models that are accurate enough for the systemic validation and involve lesser computations than 3D CFD models. In fact, the 3D CFD models are usually described by PDEs to represent complex physical phenomena and the simulation of a few seconds takes hours in a High Performance Computer (HPC).

The next level of validation is to simulate the 0D phenomenological models in the context of HIL, involving real-time constraints. However, currently it is difficult to use such high-fidelity models in HIL simulation. Conventionally, to satisfy the execution time constraints, the model must be stripped down which makes it suffering from poor representativeness (use of look-up table, quasi-static models, etc.). Moreover, to be fast enough, fixed step solvers with no error control are commonly used and preferred over error control solvers, which implies a big loss in accuracy (which is traded for execution time).

Keeping phenomenological models in a HIL simulation is an objective of this thesis to improve the prototyping and validation phases of controllers. For this aim, a first direction is to consider the use of step-size and order control solvers in the real-time context with root-finding algorithms for hybrid systems. A second direction is to provide methods that exploit efficiently the available parallelism provided by the multi-core chips. These methods deal with different aspects and levels such as decomposition methods, communication/synchronization, scheduling, execution order and extrapolation.

Generic methods have been studied as often as possible. However, in this thesis, some choices and constraints related to the context of IFP Energies nouvelles were imposed by the experimental framework. In fact, the import of the different dynamic models into the xMOD tool, for the modular co-simulation, was performed using the FMI specification. This choice was made since the aim is to find the best numerical solver among those presented in the state of the art. In fact, the specification provides important information for the solver, such as the discontinuities which are essential for root-finding algorithms use. Besides, with the FMI for model-exchange, it is possible to integrate the models with custom solvers. As the code source is accessible, the solver's functioning can be analyzed by inserting some probes at strategic places. All the proposed approach were based on the FMI 1.0 specification.

Since the focus is on the system level, and the target implementation plateform is the xMOD tool, the interesting kind of parallelization is across the model (system) and not across the method (solver/equation level). Moreover, the interest in parallelization across the method was not an option anyway, because it implies the ability to compute for example the different states' derivatives at the same time, which is not supported by the FMI specification. Finally, modeling nonlinear hybrid dynamical systems means intermixed equations (conditions, loop, implicit, etc.), which makes difficult their separation, then their parallelization.

Starting from a complex system that is already seen as a system of simpler systems (e.g. a powertrain consists in engine, transmission, battery, etc.), it is interesting to exploit more efficiently the available cores. The simpler systems can be also split, in their turn, into more simpler systems (e.g. the engine is split into 5 components described in chapter 8).

The gain of the system splitting is expected to be very promising at this level, since the major cost in numerical integration lies in the computation of the time derivatives and in the events detection and location (root-finding). The parallel execution of the different components can then significantly speed up the simulations.

Usually, the splitting relies on a domain-specific knowledge. Our objective is to apply the know-how of engineers and researchers of IFP Energies nouvelles about an internal combustion engine case study, then to propose partitioning methods intended for systems where finding an effective decomposition is difficult to guess, and finally propose efficient execution schemes for the partitioned simulation.

Part III

Contributions in the context of IFP Energies nouvelles

Foreword

This part presents all the proposed approaches and contributions of this thesis in the context of IFP Energies nouvelles. *First, the case study that structure the major thesis work is presented. Then, the splitting methodology from a physical point of view is described and applied directly on the case study. After that, theoretical evaluations of simulation errors are analyzed in the context of the modular co-simulation and a method of model decomposition based on the block-diagonalization of incidence matrices are proposed. Finally, two complementary methods that reduce the simulation errors while increasing the execution speed-ups are exposed. The first relies on a refined scheduling between the inputs and outputs of the models, then the second relies on a based-context extrapolation of the models' inputs.*

Chapter 7

Internal combustion engine case study

7.1 Introduction

This thesis is structured around a case study developed in IFP Energies nouvelles. The case study targets the real-time co-simulation of a gasoline engine.

The interest is to study the limits and the impact of the currently used configurations (solvers, parallelization, decomposition, etc.) on the real-time simulation of complex numerical models in order to suggest new alternatives to overcome the limits and to improve and extend some selected state of the art methods.

7.2 Case study overview

The considered cyber-physical system involves a Spark Ignition engine and its controllers (see figure 7.1). The engine represents the physical system part, it is modeled in the continuous-time domain using hybrid ODEs. It belongs to hybrid systems category because of some discontinuous behaviors that correspond to events triggered off when a given threshold is crossed. Controllers, which interact with physical parts, are computational devices. Controllers are detailed in [1].

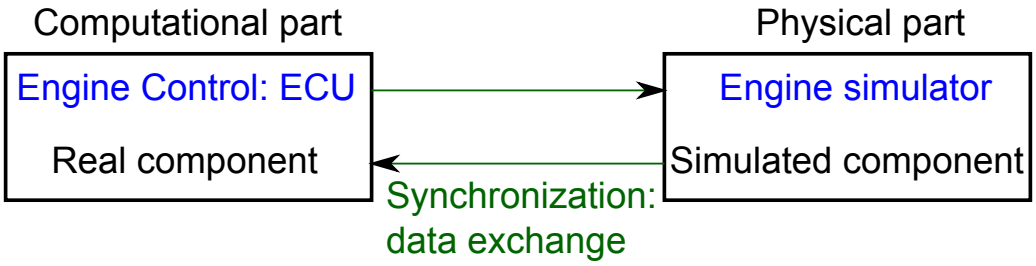

Figure 7.1: CPS: ECU + Engine model.

7.3 Spark Ignition engine functioning

The Spark Ignition (or gasoline) engine is an internal combustion engine. The mechanical work is produced by the combustion of a fuel mixture inside a cylinder, which moves a piston up

and down. The two extremes of motion are respectively called Top Dead Center (TDC) and Bottom Dead Center (BDC). The angle between the connecting rod and the piston changes when the rod moves up and down and so rotates around the crankshaft. The crankshaft angle, denoted α, is a reference signal shared by all the components (another globally shared variable is time).

The essential features of the engine lie in its feed and combustion modes. Indeed, the engine is supplied with an air-fuel mixture. The amount of admitted air is modulated by a component located in the tubing admission (the throttle) and the fuel mass is determined by a carburetor or an injection system.

The combustion of a gasoline engine requires a spark-ignition system instead of compression heating ignition system. The functioning of a gasoline engine with four-stroke is based on the "Beau de Rochas" cycle (see figure 7.2).

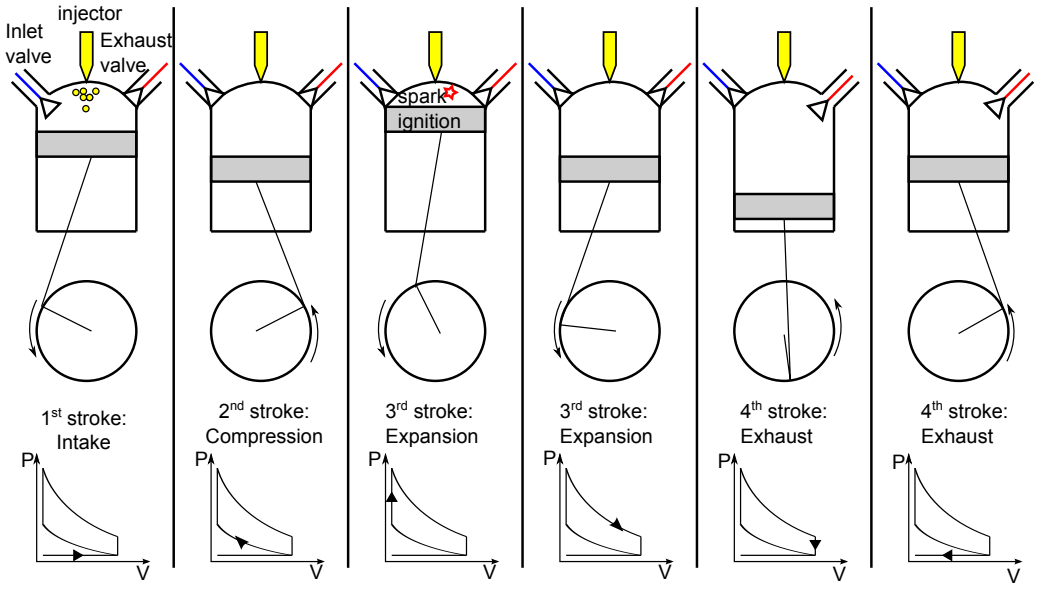

Figure 7.2: Gasoline engine functioning.

Each cylinder requires 4 strokes (engine cycles) of its piston (two revolutions of the crankshaft) to complete the sequence of events which produces one power stroke. It comprises [96]:

Intake stroke $(0° < \alpha < 180°)$

The intake stroke, which draws up fresh mixture of fuel and air (indirect injection) into the cylinder from the inlet valve, occurs when the piston moves down from TDC to BDC, creating a low-pressure area in the cylinder. Regardless of the inlet pressure in the intake system, the amount of allowed mixture depends on the throttle opening, which also determines the pressure upstream the valves. The intake pressure is very dependent on the engine speed and the engine load (torque output of the engine).

Compression stroke $(180° < \alpha < 360°)$

The compression stroke occurs when both valves are closed and the piston moves up from BDC to TDC creating a compression of the mixture inside the cylinder to a small fraction of its initial

volume. Just before the end of the compression stroke, combustion is initiated by a spark and the cylinder pressure rises very fast.

Power or expansion stroke $(360° < \alpha < 540°)$

The power stroke or the expansion stroke (combustion phase) occurs when the piston is close to TDC and the compressed air-fuel mixture is ignited by a spark plug. This leads to a high-temperature and high-pressure gases which push the piston down to BDC and so force the crankshaft to rotate. The gas pressure and the gas temperature decrease during the supplied work to the piston.

Exhaust stroke $(540° < \alpha < 720°)$

An exhaust stroke occurs when the piston reaches BCD, the remaining burned gases exit the cylinder through the exhaust valve as a result of their own pressure, then under the push of the piston.

7.4 Spark Ignition F4RT engine

7.4.1 Engine description

In this study, a Spark Ignition (SI) RENAULT F4RT engine (see figure 7.3) has been modeled with 3 gases (air, fuel and burned gas). It is a four-cylinder, in line Port Fuel Injector (PFI), engine in which the engine displacement is 2 L. The combustion is considered as homogeneous. The air path uses a turbocharger with a mono-scroll turbine controlled by a waste-gate, an intake throttle and a downstream-compressor heat exchanger. To finish, this engine is equipped with two Variable Valve Timing (VVT) devices, for intake and exhaust valves, to improve the engine efficiency (performance, fuel and emissions). The maximum power is about 136 kW at 5000 rpm.

Figure 7.3: RENAULT F4RT.

7.4.2 Combustion model description

Two different types of combustion models have been used in this study, the Wiebe and CFM models. These models share basic thermodynamic equations, as the equation of mass conservation, the ideal gas equation, the equation of energy conservation, etc. The principle difference between these models is in terms of heat exchange during the combustion i.e. how the combustion is maintained.

Wiebe model

The Wiebe model is an empirical model for combustion heat released [97] that presents much less complexity than the CFM model. It is based on a mix of physical approaches and identifications or learning processes, applied on the results of an experimental or/and numerical combustion campaign performed with a more complex model.

The combustion heat release Q_{comb}, which is function of the crankshaft angle α, is defined as follow:

$$\frac{dQ_{\text{comb}}}{d\alpha} = \frac{Q_{\text{tot}}}{\Delta\theta} A_1(1+f_1)y^{f_1}\exp(-A_1 y^{(1+f_1)}),$$

where Q_{tot} is the total energy of the fuel that is equal to:

$$Q_{\text{tot}} = \begin{cases} m_{\text{fuel}}.\text{LHV}.\eta & if \text{AFR} < 1, \\ \\ \frac{m_{\text{air}}}{\text{Pco}}.\text{LHV}.\eta & otherwise. \end{cases}$$

AFR is the Air Fuel equivalence Ratio, A_1 and f_1 are constants, η is the combustion efficiency and LHV is the Lower Heating Value of the fuel. Pco is the combustive power of the fuel that corresponds to the ratio between the mass of air m_{air} and the mass of fuel m_{fuel} during a complete combustion. θ_0 is the initial combustion angle, Δ_θ corresponds to the duration of burn and y is a ratio defined as follow:

$$y = \frac{\alpha - \theta_0}{\Delta_\theta}.$$

The main advantage of this model is to take into account the behavior of the engine with a crankshaft angle degree time-scale, which is not the case of look-up table models.

In terms of complexity, the Wiebe-based engine model has 78 continuous states $\boldsymbol{X}$, 420 event indicators (of discontinuities) $\boldsymbol{Z}$, 1334 equations and 7767 variables (including 1922 unknowns).

CFM model

The phenomenological CFM (Coherent Flame Model) is a 0D model described by ODEs and developed by IFP Energies nouvelles [98]. It is based on the reduction of the 3D CFD model described by PDEs and called ECFM (Extended Coherent Flame Model) model [99].

The CFM model is intended to be more predictive than the Wiebe model since the heat release during combustion is described through physical equations. Its formalism distinguishes two zones: a zone of burned gas and a zone of unburned gas. The flame propagates from the

burned gas to the unburned gas and the chemical reactions of fuel oxidation occur in a very thin layer.

The reduction of the different equations, from 3D to 0D, were performed following assumptions:

- The burned and unburned gases are considered as ideal gases;
- The temperature and the mixture composition are considered homogeneous in both zones;
- The pressure is considered the same in both zones.

The rate of heat release during combustion $\dot{Q}_{comb}$ is calculated as follows:

$$\dot{Q}_{comb} = \mathrm{LHV}.\dot{\omega}_F,$$

where $\dot{\omega}_F$ is the fuel consumption rate computed as follow:

$$\dot{\omega}_F = \rho_{fg} Y_{fg}^f U_l S_f, \tag{7.1}$$

with ρ_{fg} is the density of the unburned gas, Y_{fg}^f is the mass fraction of unburned gas, U_l is the laminar flame speed and S_f is the laminar flame surface.

In this model, the rate of fuel consumption defined in (7.1) depends on the flame surface, computed thanks to the laminar flame speed and the turbulent kinetic energy. Only one parameter related to turbulent kinetic energy is tuned for combustion calibration. The other ones remain constant for the whole operating conditions.

The CFM-0D model is the typical modeling level able to combine a good representation of physical phenomena with reasonable CPU performances. Thanks to these characteristics, this model can be embedded in a full engine simulator and used for architecture design or control strategy development issues [100].

In terms of complexity, the CFM-based engine model has 118 continuous states $\boldsymbol{X}$, 398 event indicators (of discontinuities) $\boldsymbol{Z}$, 1466 equations and 7907 variables (including 1979 unknowns).

7.5 Engine modeling and simulation

7.5.1 Engine modeling

The F4RT engine model, as illustrated in figure 7.4 was developed using ModEngine library [101]. ModEngine is a Modelica [102] library that allows the modeling of a complete engine with diesel and gasoline combustion models.

Requirements for the ModEngine library were defined and based on the already existing IFP-Engine library. The development of the IFP-Engine library was performed several years ago at IFP Energies nouvelles and it is currently used in the AMESim tool.

The ModEngine contains more than 250 sub models. It has been developed to allow the simulation of a complete virtual engine using a characteristic time-scale based on the crankshaft angle. A variety of elements are available to build representative models for engine components, such as turbocharger, wastegate, gasoline or Diesel injectors, valve, air path, Exhaust Gas Recirculation (EGR) loop etc. ModEngine is currently functional in the Dymola tool.

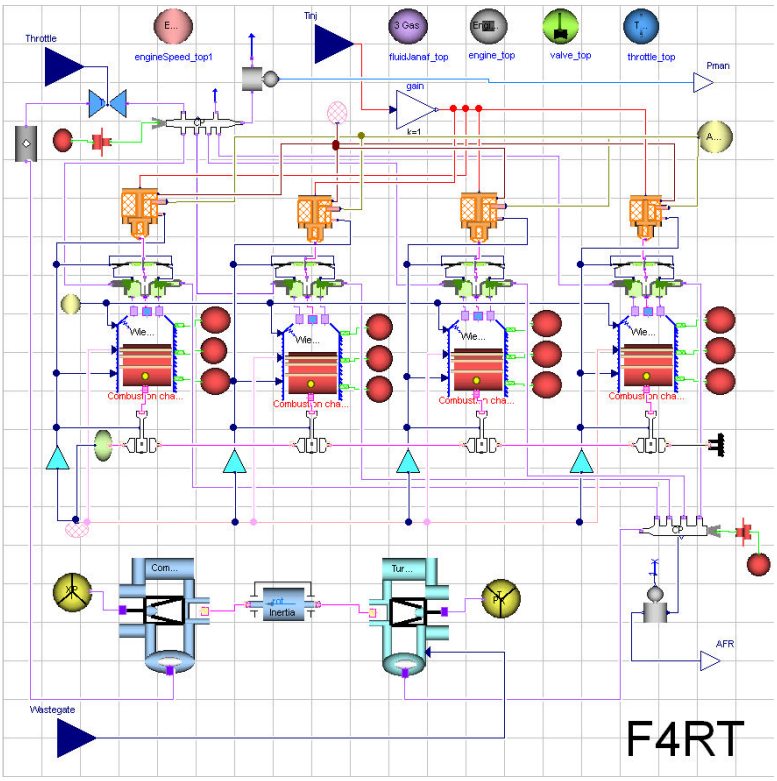

Figure 7.4: F4RT engine modeled in Dymola.

7.5.2 Engine simulation

The engine model was imported into xMOD as shown in figure 7.5 using the FMI [103] export features of Dymola. Specifically, the FMI standard describes the software interface of a hybrid ODE system. Then, the engine model is linked to its controller developed in Simulink thanks to the integration capabilities of xMOD.

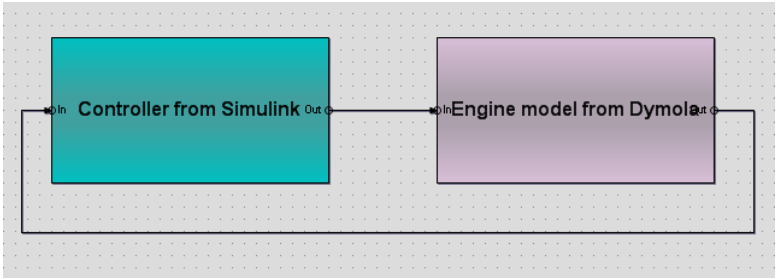

Figure 7.5: F4RT engine model exported in xMOD.

7.6 Events origin in the engine model

The engine model is described using ODEs where the state vector $\boldsymbol{X}$ comprises the crankshaft angle, the energy, the temperature, the mass of the three gases, etc. Furthermore, the model is described also by some discontinuities $\boldsymbol{Z}$ related to events happening during the engine cycle. These events are the spark angle, the injection reach, the start of combustion, etc.

The reduction from 3D to 0D model makes simulation tools efficient, especially in the combustion chamber where the combustion and pollutant formation processes take place. However it generates additional "non-physical" artifact events related to constraints. For this reason, the used engine model presents many events that can be classified as follow:

- Real physical events, e.g. spark advance time, engine cycle, intake valve lift, exhaust valve lift;
- Trigger events: Conditions that trigger the real physical events. For example, the event engine cycle depends on some thermodynamic condition, triggered by thresholds on the values of intake and exhaust mass flows;
- Mathematical exception handling: Conditions that avoid the division by zero, the square root of a negative value, etc.;
- Other events: Some conditions can be duplicated several times just for coding clarity. For example, when a condition implies many assignments of variables that are used in different parts of the code, the condition is duplicated to bring closer each variable to the location where it is usually used.

7.7 Conclusion

Nowadays, real-time simulation is increasingly needed and used in the automotive domain. In fact, for the European emission standard "Euro 6", tests on engine testbeds must be normalized and follow the WLTC[1] driving cycles[2] to determine the quantity of fuel consumption and polluting emissions. For this aim, engineers are forced to spend time in converting phenomenological models to Look-up table static models. This internal combustion engine model presents a good case study that describes the HIL problematic. Our objective is to present methods that improve the computation time of this model towards a real-time simulation while preserving the 0D model fidelity and keeping accuracy under control.

[1] World-wide harmonized Light duty driving Test Cycle
[2] Driving cycles are sequences of set points concerning the vehicle speed.

Chapter 8

Model decomposition from a physical point of view

8.1 Introduction

This chapter describes the methodology in splitting a system into several subsystems based on the knowledge of the physical system behavior. This approach is applied on the Spark Ignition F4RT engine model (previously described in chapter 7) and it can be reproduced on a different complex hybrid dynamical systems, based of course a good knowledge of the physical system functioning.

8.2 Computational decomposition approach

In the systemic approach, the complex system is seen as a set of subsystems. Since our approach is interested on the thread-level parallelism, to link this approach to the physical system, each subsystem is mapped to a thread as it shown in figure 8.1. The connections between the subsystems represent the different data flow, exchanged between them. From the computing tasks viewpoint, these edges or dependencies define the execution order between the threads (represented by the nodes).

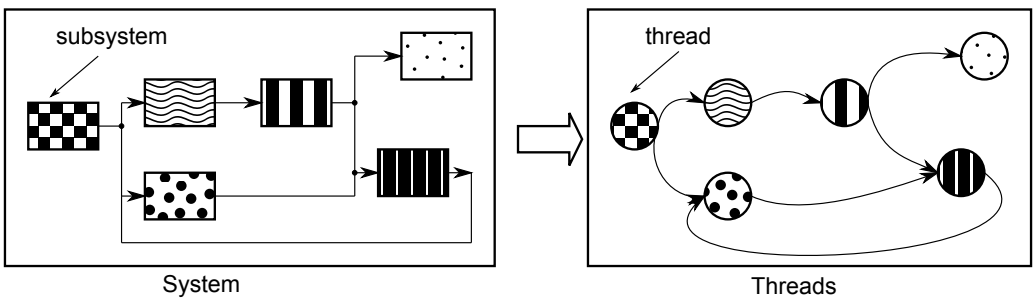

Figure 8.1: Mapping of the subsystems to the threads.

8.3 Model decomposition approach

The natural and intuitive partitioning of the engine model is performed by separating the four cylinders from the air path (AP), then by isolating the cylinders (C_i, for $i \in [1,2,3,4]$) from each other.

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.

The dynamics of the air path is slow (it produces slowly varying outputs to the cylinders, e.g. temperature) compared to those of the cylinders (they produce fast outputs to the air path, e.g. torque). Besides, unlike the cylinders outputs, most air path outputs are not a direct function of the air path inputs (they are called Non Direct Feedthrough (NDF) outputs and defined in section 9.2.1). This results in choosing the execution order of the split model from the air path to the cylinders (in accordance with the analysis of the behavior of Non Direct Feedthrough (NDF) to Direct Feedthrough (DF) described in section 9.3.2).

The model is divided into 5 components and governed by a basic controller denoted CTRL as it is shown in figure 8.2. It gathers 91 inputs and 98 outputs regardless of the chosen combustion model (the Wiebe model or the CFM model).

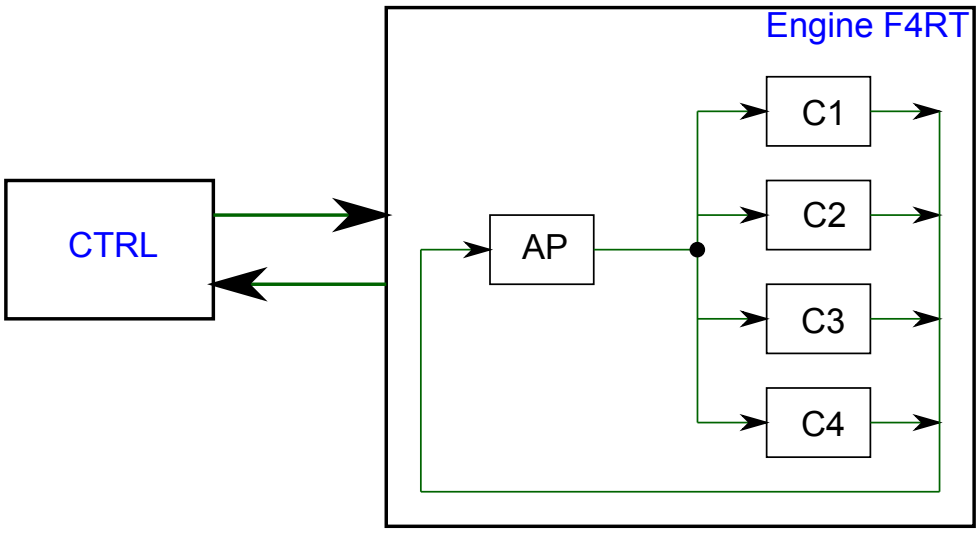

Figure 8.2: Split engine model.

8.4 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 slow to fast models. For the case study, all the cylinders must wait for the execution of the air path;

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

These methods are sketched in figure 8.3, where DT is the execution time during a communication step. DT gathers the integration of the models in the blocks X_i (e.g. X_{AP} for air path AP) and the input and output updates in the IN/OUT blocks.

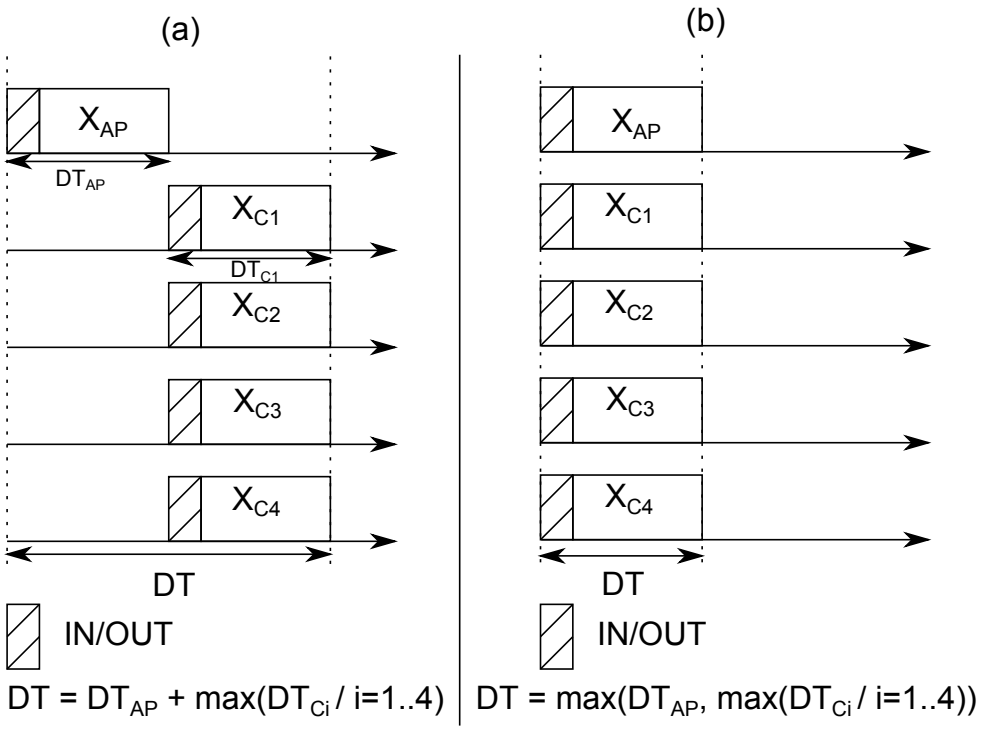

Figure 8.3: (a) sv-MCosim method; (b) ev-MCosim method.

8.5 Test results

In the following tests, simulations of the F4RT engine are done in xMOD for both Wiebe and CFM combustion models. In a first approach, the idea is to compare the variable-step solver LSODAR against the fixed-step solver RK4 with a small integration step-size (50 µs), considered as a reference by model developers. The validation will be performed using the quantities of interest as intake and exhaust manifold pressures, AFR and torque.

Before using the LSODAR solver locally in each subsystem (thread), an important preliminary work is performed to integrate LSODAR in the FMI for Model Exchange framework by making it thread safe[1].

[1]An implementation or a piece of code is thread safe when the manipulation of shared data structures is guaranteed to be free of race conditions (safe execution) when accessed by multiple threads simultaneously.

8.5.1 Accuracy with variable-step solver

Figure 8.4 shows the intake manifold pressure and the torque during 1 engine cycle corresponding to 2 crankshaft revolutions (using a Wiebe model with an engine speed equal to 2500 rpm). These outputs are computed using both LSODAR with a communication time-step equal to 500 µs and a tolerance equal to 10^{-5} and RK4 with a time-step equal to 50 µs, where the accuracy is ensured. In fact, the error between the outputs of manifold pressure is less than 0.3 % and for the torque is less than 0.5 %.

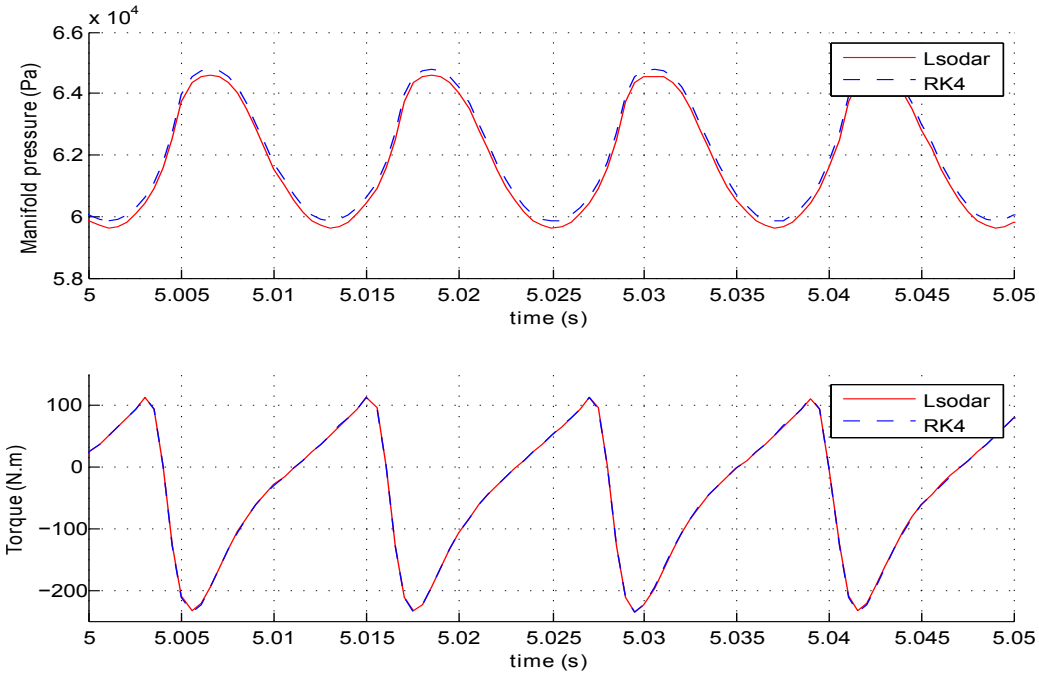

Figure 8.4: Several outputs using RK4 and LSODAR solvers.

With variable-step solver, the bounding of the error due to the integration is ensured. However, at the same time, the execution time is 4 times longer with a tolerance equal to 10^{-4} and 6 times longer with a tolerance equal to 10^{-5}.

After deeply analyzing the solver's execution, the slowness may be explained by the presence of a large number of discontinuities that decreases the speed advantage of variable-step solvers. In fact, discontinuities involve a costly computation of the *zero-crossing* function in (3.6), used for *events detection* and *location*, then a restart of the solver for *events handling*.

Since the events are related usually to the evolution of a subset of the state vector, the partitioning of the engine model is performed from a physical point of view as described in section 8.3, so that each subsystem can be integrated by its own solver, avoiding interrupts coming from unrelated events. In fact, the combustion phase raises most of the events, 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.

8.5.2 Model splitting effect on execution time with single-core

The first step is to compare the execution time between the original model and the split model but executed on a single-core, to only check the effect of events relaxation on the speed-up of LSODAR solver without the effect of the parallelization.

Result 1: Number of discontinuities

The partitioning of the model involved the decrease of the number of the discontinuities seen by the solver. In fact, tests during 0.3 s show that the unpartitioned model presents 851 events whereas the split model presents on average 203 events per cylinder and 119 for the air path. Figure 8.5 summarizes that during 2 engine cycles.

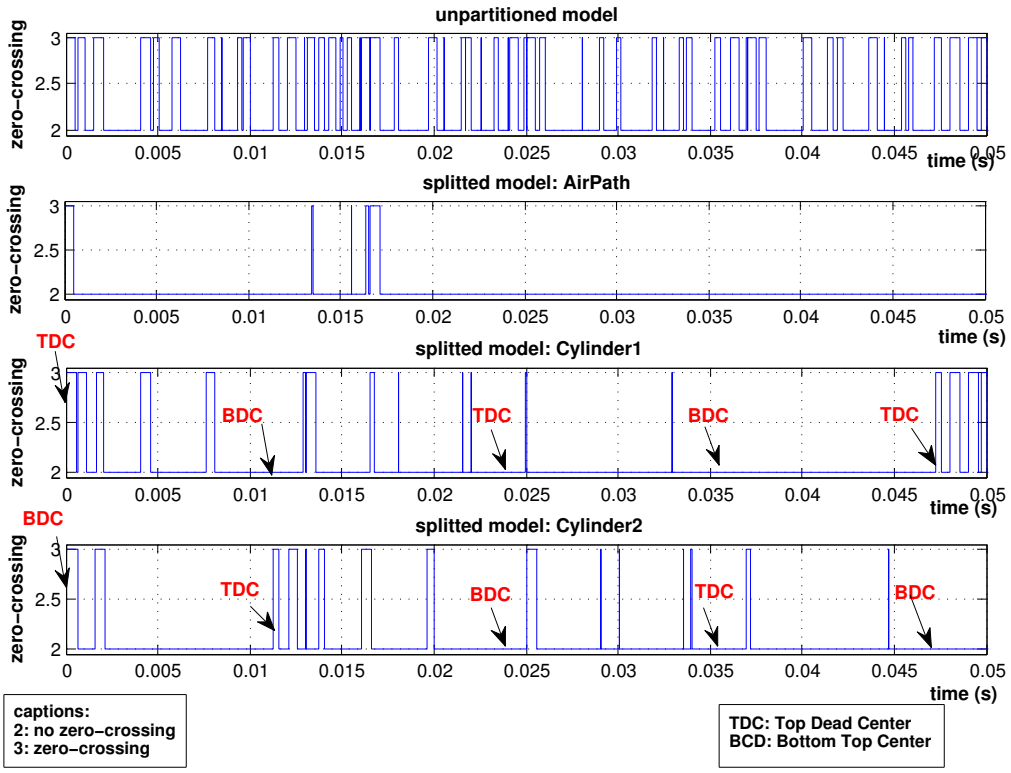

Figure 8.5: Number of discontinuities per model.

Result 2: Integration step-size

The impact of events reduction per subsystems involves the decrease of the number of integration interrupts, so the increase of the time-step size as shown in figure 8.6. For the global model, the maximum and the mean value of the step-size are around $h_{\max} = 422\,\mu s$ and $h_{\text{mean}} = 148\,\mu s$ whereas for the split model, the step-size reaches the maximum allowed one $h_{\max} = 500\,\mu s$ and the mean value is around $h_{\text{mean}} = 215\,\mu s$ for the cylinders and $h_{\text{mean}} = 229\,\mu s$ for the air path.

Result 3: Execution time

Results 1 and 2 entail a speed-up of the execution time, about 1.98 without the use of multi-core parallelization.

8.5.3 Model splitting effect on execution time with multi-core

In this section, the interest is on the parallelization of the model using a multi-core PC. Tests are done using RK4 solver on a CFM model. The engine model is executed as shown in figure 8.7

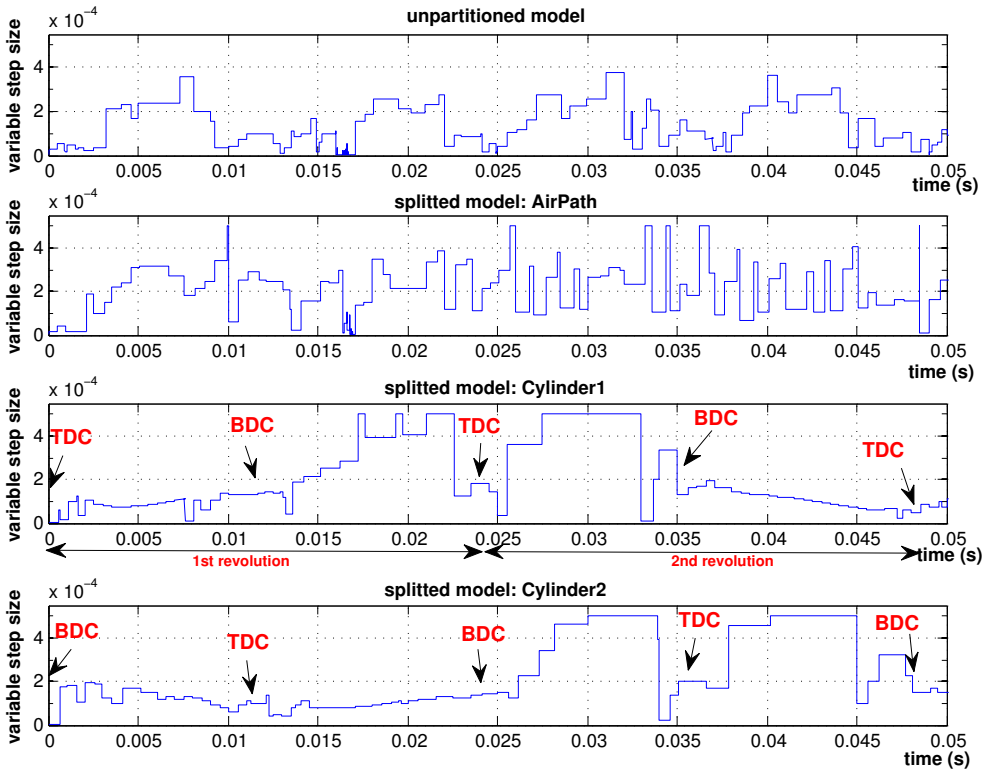

Figure 8.6: Integration step behavior per model.

using at first 2 cores then 4 cores. After that, the inter-subsystem dependencies are ignored and the model is executed on 5 cores.

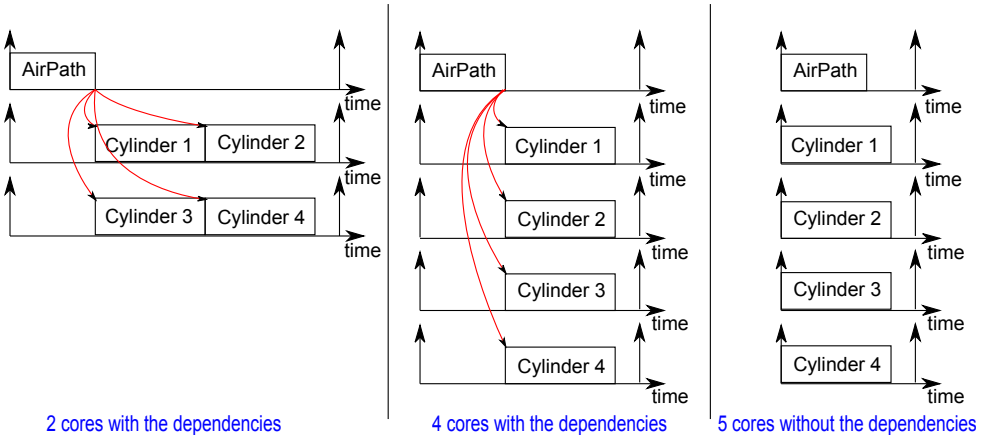

Figure 8.7: Distribution of the engine model.

Test results show that the speed-up of execution time, compared with single-core, is about 1.77 when using 2 cores, then it is increased to 3.15 when using 4 cores. For the last case, the speed-up is about 3.9. The question then is "what is the best trade-off regarding execution time and accuracy: relaxing these dependencies and running the model at 50 µs, or keeping them and running it with 100 µs".

Table 8.1 shows that ignoring the dependencies between the air path and the four cylinders and using RK4 with a time-step $h = 50\,\mu s$ presents less error in major outputs than keeping the dependencies but integrating with a time-step $h = 100\,\mu s$.

Table 8.1: Error on several outputs.

	Case 1[2]:Er(%)	Case 2[3]:Er(%)
AFR	1.02	0.53
Intake Pressure	0.47	0.42
Compressor Speed	0.65	0.50
Exhaust Pressure	0.93	0.94
Torque	7.32	0.55

Regarding the execution time, the second case is faster than the first one, the speed-up is around 2.06. We can conclude then that executing the F4RT engine with the CFM model under a multi-core machine is better in term of execution time and accuracy, when the dependencies between the air path and the cylinders are ignored with a time-step equal to 50 µs, than when the dependencies are respected with a time-step equal to 100 µs.

These results show the importance and the impact of the choice on how and where tuning some parameters (e.g. integration step, communication step, model of computations, etc.) may affect the simulation result accuracy. It leads to the necessity to evaluate the simulation error and to analyze the convergence of the results. A hint, thanks to these current results, makes us to firstly think about the communication step.

8.6 Conclusion

The current study showed that decoupling the model parts by relaxing their data dependencies is promising in term of simulation speed (by increasing the parallelism) and even in results accuracy with an adequate choice on the communication step. This method presents an important potential to improve the simulation of complex systems.

Besides, tests results on engine model showed that, with the model partitioning, it is possible to efficiently use variable-step solvers thanks to the decrease of the number of discontinuities, so the number of integration interrupts, in each subsystem.

The use of variable-step solvers in parallel modular co-simulation approach improves the simulation time. It keeps also the results accuracy under control, by bounding locally (in the subsystem) the integration error. The next chapter investigates on the global error evaluation, regardless the case study and the kind of used numerical solver, in the context of the modular co-simulation.

[2] sv-MCOSIM and h=100 µs
[3] ev-COSIM and h=50 µs

Chapter 9

Error evaluation for modular co-simulation

9.1 Motivation for multi-simulator approach

9.1.1 Efficient handling of discontinuities

Complex physical systems are generally modeled by hybrid nonlinear 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 [35].

As it was shown in chapter 8, the numerous discontinuities in the hybrid system sadly prevent variable step solvers to reach the high integration speeds which could be attained only considering the system's continuous dynamics. In addition, by integrating each subsystem by its own solver thanks to the FMI specification, interrupts coming from unrelated events are avoided and events detection and location inside a subsystem are processed faster because they involve a smaller variables set.

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

9.1.2 Model formalization for the modular co-simulation

To execute the system in parallel, the initial hybrid dynamical system Σ' described in (3.5) is split into several sub-models.

For simplicity, assume that the system is decomposed into two separate blocks denoted model

1 and model 2, in figure 9.1. Our approach generalizes to any decomposition into B blocks of system Σ', $(b = 1, \ldots, B)$.

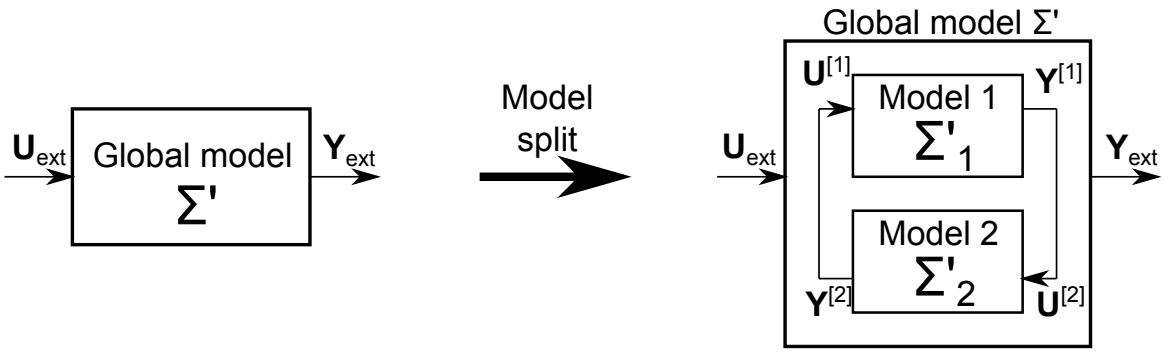

Figure 9.1: System splitting for parallelization.

Therefore, the subsystems can be written as:

$$\left\{ \begin{array}{rcl} \dot{\boldsymbol{X}}^{[1]} & = & \boldsymbol{f}^{[1]}(t, \boldsymbol{X}^{[1]}, \boldsymbol{D}^{[1]}, \boldsymbol{U}^{[1]}, \boldsymbol{U}_{ext}) \\ \boldsymbol{Y}^{[1]} & = & \boldsymbol{g}^{[1]}(t, \boldsymbol{X}^{[1]}, \boldsymbol{D}^{[1]}, \boldsymbol{U}^{[1]}, \boldsymbol{U}_{ext}) \end{array} \right. \quad and \quad \left\{ \begin{array}{rcl} \dot{\boldsymbol{X}}^{[2]} & = & \boldsymbol{f}^{[2]}(t, \boldsymbol{X}^{[2]}, \boldsymbol{D}^{[2]}, \boldsymbol{U}^{[2]}, \boldsymbol{U}_{ext}) \\ \boldsymbol{Y}^{[2]} & = & \boldsymbol{g}^{[2]}(t, \boldsymbol{X}^{[2]}, \boldsymbol{D}^{[2]}, \boldsymbol{U}^{[2]}, \boldsymbol{U}_{ext}) \end{array} \right. \tag{9.1}$$

with $\boldsymbol{X} = [\boldsymbol{X}^{[1]}\ \boldsymbol{X}^{[2]}]^T$ and $\boldsymbol{D} = [\boldsymbol{D}^{[1]}\ \boldsymbol{D}^{[2]}]^T$, where ^{T} denotes the matrix transpose.

Here $\boldsymbol{U}^{[1]}$ are the inputs needed for model 1 (Σ'_1), directly provided by the outputs $\boldsymbol{Y}^{[2]}$ produced by model 2 (Σ'_2). Similarly, $\boldsymbol{U}^{[2]}$ are the inputs needed for model 2 directly provided by the outputs $\boldsymbol{Y}^{[1]}$ produced by model 1.

To perform the numerical integration of the whole multi-variable system, each of these simulators needs to exchange, at communication (or synchronization) points t_{s_b}, the data needed by the others (in figure 9.2, $b = 1, 2$).

To speed up the integration, the parallel branches must be as independent as possible, so that they are synchronized at a rate $H^{[b]} = t_{s_b+1} - t_{s_b}$ by far slower than their internal integration step $h^{[b]}_{n_b}$ ($H^{[b]} \gg h^{[b]}_{n_b}$). Therefore, between communication points, each simulator integrates at its own rate (assuming a variable step solver), and considers that the data incoming from others simulators is hold as constant.

It is likely that large communication intervals allow to speed up the numerical integration, but may result in integration errors and poor confidence in the final result. For example, [104] studied the trade-off between the stability (or accuracy) and the computational performances in the context of the modular co-simulation of strongly coupled systems. Modeling the errors induced by slack synchronization is a first step to find effective directions to improve the trade-off between integration speed and accuracy.

In a preliminary approach for error evaluation, it is assumed that a common communication step-size $H = t_{s+1} - t_s$ is shared by all blocks ($H^{[b]} = H$, for $b = 1, \ldots, B$), so that they all read their inputs and update their outputs at communication points that are multiple of H. Then, all the results about error evaluation will be generalized for a multi-rate co-simulation in section 9.5.

For the sake of simplicity, the theoretical and analytical error evaluation will consider the system's solution steady and regular enough, regarding the discontinuities' effects. The assumption is to neglect the discontinuities at this study level and take it into account at the simulation level (numerical solvers, root-finding algorithm, extrapolation, etc.).

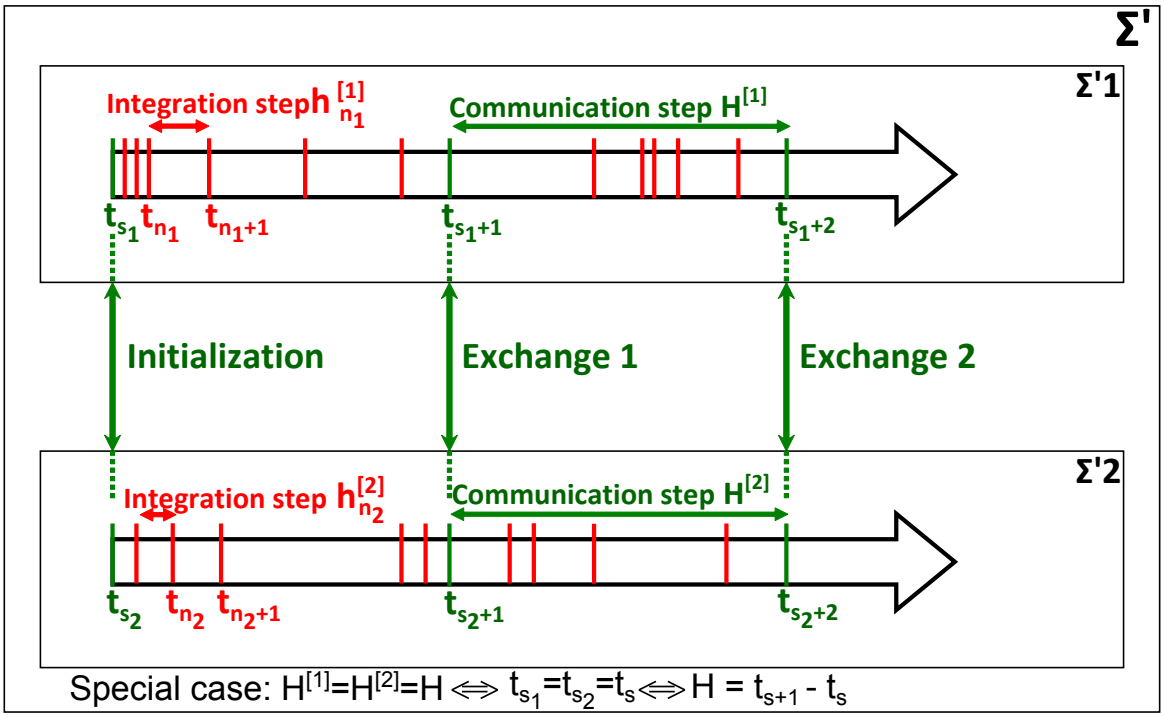

Figure 9.2: Σ' split into Σ'_1 and Σ'_2 for parallel simulation.

Besides, the numerical round-off errors, induced by limited floating point precision of the calculator, are not taken into account in the following analysis.

9.2 Error evaluation and convergence analysis for the sequential modular co-simulation

The sequential modular co-simulation of the split model represents the modular co-simulation performed with respect to data dependencies. This is the standard version of the modular co-simulation, denoted "sv-MCosim", where the execution order is fixed.

9.2.1 Bound of the global error on the states

We consider here, as a first approach, that the model splitting does not bring loops and that the execution order between sub-models is defined naturally. The study around the loops will be performed later in section 9.3.

The aim is to generalize the evaluation of the local and global integration errors ($\boldsymbol{\delta}_n$ and $\boldsymbol{\Delta}_n$) performed in section 4.2.3 which is based on the simple system defined in (3.1) to another evaluation based on the hybrid dynamical system described in (3.5) and the split model defined in (9.1).

This means that the old $\boldsymbol{\Phi}^{[b]}(\boldsymbol{X}^{[b]}(t_n), \boldsymbol{f}^{[b]})$ is considered now function of the inputs $\boldsymbol{U}_{ext}$, $\boldsymbol{U}^{[b]}$ and the discrete states $\boldsymbol{D}^{[b]}$ too, namely $\boldsymbol{\Phi}^{[b]}(\boldsymbol{X}^{[b]}(t_n), \boldsymbol{D}^{[b]}(t_n), \boldsymbol{U}^{[b]}(t_n), \boldsymbol{U}_{ext}(t_n), \boldsymbol{f}^{[b]})$.

It is clear that the index n is actually n_b, because it is related to the model b (each model varies according to its own integration rate). However the term n_b will be only mentioned when there is a risk of confusion or misunderstanding.

For the forthcoming evaluation, the external inputs $\boldsymbol{U}_{ext}$ and the discrete states $\boldsymbol{D}^{[b]}$ are omitted for clarity, that is $\boldsymbol{\Phi}^{[b]}(\boldsymbol{X}^{[b]}(t_n),\boldsymbol{U}^{[b]}(t_n),\boldsymbol{f}^{[b]})$.

Reminding that

$$\boldsymbol{\Delta}_{n+1}^{[b]} = \boldsymbol{X}^{[b]}(t_{n+1}) - \boldsymbol{X}_{n+1}^{[b]},$$

$\boldsymbol{\Delta}_{n+1}^{[b]}$ satisfies the same relationship found in (4.15):

$$\boldsymbol{\Delta}_{n+1}^{[b]} = \boldsymbol{\Delta}_{n}^{[b]} + h_n\left(\boldsymbol{\Phi}^{[b]}(\boldsymbol{X}^{[b]}(t_n),\boldsymbol{U}^{[b]}(t_n),\boldsymbol{f}^{[b]}) - \boldsymbol{\Phi}^{[b]}(\boldsymbol{X}_n^{[b]},\boldsymbol{U}_n^{[b]},\boldsymbol{f}^{[b]})\right) + \boldsymbol{\delta}_{n+1}^{[b]}. \tag{9.2}$$

Now adding and subtracting the same term $\boldsymbol{\Phi}^{[b]}$ as follow:

$$\begin{aligned}\boldsymbol{\Delta}_{n+1}^{[b]} &= \boldsymbol{\Delta}_{n}^{[b]} + h_n\left(\boldsymbol{\Phi}^{[b]}(\boldsymbol{X}^{[b]}(t_n),\boldsymbol{U}^{[b]}(t_n),\boldsymbol{f}^{[b]}) - \boldsymbol{\Phi}^{[b]}(\boldsymbol{X}_n^{[b]},\boldsymbol{U}^{[b]}(t_n),\boldsymbol{f}^{[b]})\right)\\ &+ h_n\left(\boldsymbol{\Phi}^{[b]}(\boldsymbol{X}_n^{[b]},\boldsymbol{U}^{[b]}(t_n),\boldsymbol{f}^{[b]}) - \boldsymbol{\Phi}^{[b]}(\boldsymbol{X}_n^{[b]},\boldsymbol{U}_n^{[b]},\boldsymbol{f}^{[b]})\right) + \boldsymbol{\delta}_{n+1}^{[b]},\end{aligned} \tag{9.3}$$

and considering $\boldsymbol{\Phi}^{[b]}$ function w.r.t. arguments $\boldsymbol{X}^{[b]}$, $\boldsymbol{U}^{[b]}$ satisfy a Lipschitz condition with a Lipschitz constant L, the global error on $\boldsymbol{X}^{[b]}$ can be bounded as follow:

$$\left\|\boldsymbol{\Delta}_{n+1}^{[b]}\right\| \leq \left\|\boldsymbol{\Delta}_{n}^{[b]}\right\| + h_nL\left\|\boldsymbol{X}^{[b]}(t_n) - \boldsymbol{X}_n^{[b]}\right\| + h_nL\left\|\boldsymbol{U}^{[b]}(t_n) - \boldsymbol{U}_n^{[b]}\right\| + \max_{0\leq k\leq n+1}\left\|\boldsymbol{\delta}_k^{[b]}\right\|. \tag{9.4}$$

Using the terms h and p defined in (4.6) and (4.7) as well as the $\mathcal{O}()$ notation defined in (4.3), the bounding is transformed to

$$\begin{aligned}\left\|\boldsymbol{\Delta}_{n+1}^{[b]}\right\| &\leq (1+hL)\left\|\boldsymbol{\Delta}_{n}^{[b]}\right\| + \max_{0\leq k\leq n+1}\left\|\boldsymbol{\delta}_k^{[b]}\right\| + hL\left\|\boldsymbol{U}^{[b]}(t_n) - \boldsymbol{U}_n^{[b]}\right\|\\ &\leq \frac{(1+hL)^{(n+1)}-1}{hL}\left(\max_{0\leq k\leq n+1}\left\|\boldsymbol{\delta}_k^{[b]}\right\| + hL\max_{0\leq k\leq n}\left\|\boldsymbol{U}^{[b]}(t_k) - \boldsymbol{U}_k^{[b]}\right\|\right)\\ &\leq \frac{(e^{t_{n+1}-t_0}-1)}{hL}\max_{0\leq k\leq n+1}\left\|\boldsymbol{\delta}_k^{[b]}\right\| + (e^{t_{n+1}-t_0}-1)\max_{0\leq k\leq n}\left\|\boldsymbol{U}^{[b]}(t_k) - \boldsymbol{U}_k^{[b]}\right\|\\ &= \mathcal{O}\left(\frac{h^{p+1}}{h}\right) + \mathcal{O}(H)\\ &= \mathcal{O}(h^p) + \mathcal{O}(H).\end{aligned} \tag{9.5}$$

The global error on the states is clearly bounded by two terms. The first term is related to the applied numerical solver, more specifically the time-step and the order. The worst case scenario on the bounding would be the maximum used integration step h and order p. On the other hand, the second term is related to the size of the communication step H. However, the weight of each term on the error is deeply related to the size of the communication step regarding the integration step. Based on the same approach used in [105], it is clear that the communication step H dominates the error when $H \gg h$.

9.2.2 Bound of the global error on the outputs

The global integration error on $\boldsymbol{Y}^{[b]}$ is defined as follow

$$\boldsymbol{Y}^{[b]}(t_n) - \boldsymbol{Y}_n^{[b]} = \boldsymbol{g}^{[b]}(\boldsymbol{X}^{[b]}(t_n), \boldsymbol{U}^{[b]}(t_n)) - \boldsymbol{g}^{[b]}(\boldsymbol{X}_n^{[b]}, \boldsymbol{U}_n^{[b]}).$$

Based on the same strategy, adding and subtracting the same term of $\boldsymbol{g}^{[b]}$, the global error becomes

$$\begin{aligned} \boldsymbol{Y}^{[b]}(t_n) - \boldsymbol{Y}_n^{[b]} &= \left(\boldsymbol{g}^{[b]}(\boldsymbol{X}^{[b]}(t_n), \boldsymbol{U}^{[b]}(t_n)) - \boldsymbol{g}^{[b]}(\boldsymbol{X}_n^{[b]}, \boldsymbol{U}^{[b]}(t_n))\right) \\ &+ \left(\boldsymbol{g}^{[b]}(\boldsymbol{X}_n^{[b]}, \boldsymbol{U}^{[b]}(t_n)) - \boldsymbol{g}^{[b]}(\boldsymbol{X}_n^{[b]}, \boldsymbol{U}_n^{[b]})\right). \end{aligned} \tag{9.6}$$

Considering that $\boldsymbol{g}^{[b]}$ function w.r.t. arguments $\boldsymbol{X}^{[b]}$, $\boldsymbol{U}^{[b]}$ satisfy a Lipschitz condition with a Lipschitz constant L, the global error on $\boldsymbol{Y}^{[b]}$ can be bounded as follow:

$$\begin{aligned} \left\|\boldsymbol{Y}^{[b]}(t_n) - \boldsymbol{Y}_n^{[b]}\right\| &\leq L\left\|\boldsymbol{X}^{[b]}(t_n) - \boldsymbol{X}_n^{[b]}\right\| + L\left\|\boldsymbol{U}^{[b]}(t_n) - \boldsymbol{U}_n^{[b]}\right\| \\ &\leq L\left\|\boldsymbol{\Delta}_n^{[b]}\right\| + L \max_{0\leq k\leq n}\left\|\boldsymbol{U}^{[b]}(t_k) - \boldsymbol{U}_k^{[b]}\right\| \\ &= (\mathcal{O}(h^p) + \mathcal{O}(H)) + \mathcal{O}(H). \end{aligned} \tag{9.7}$$

Reminding some operations on $\mathcal{O}()$:

- f and g are functions of a variable x such that $f(x) = \mathcal{O}(g(x))$ means that $|f(x)| \leq C|g(x)|$ with $C > 0$ constant;
- *Multiplication by a nonzero real*: if $f = \mathcal{O}(g)$ and $\lambda \neq 0$ then $f = \mathcal{O}(\lambda g)$;
- *Linear combination of negligible functions regarding the same function*: if $f_1 = \mathcal{O}(g)$ and $f_2 = \mathcal{O}(g)$ then $\forall \lambda_1, \lambda_2 \in \mathbb{R}$, $\lambda_1 f_1 + \lambda_2 f_2 = \mathcal{O}(g)$;
- *Product*: if $f_1 = \mathcal{O}(g_1)$ and $f_2 = \mathcal{O}(g_2)$ then $f_1 f_2 = \mathcal{O}(g_1 g_2)$.

then (9.7) can be transformed to

$$\left\|\boldsymbol{Y}^{[b]}(t_n) - \boldsymbol{Y}_n^{[b]}\right\| = \mathcal{O}(h^p) + \mathcal{O}(H). \tag{9.8}$$

The global error on the outputs satisfies the same rule concluded for the global error on the states. The choice of large communication interval H is important to improve the trade-off between integration speed and the results accuracy.

Moreover, it is important to mention that the outputs $\boldsymbol{Y}^{[b]}$ is not always function of $\boldsymbol{U}^{[b]}$. In fact, a model is said Non Direct Feedthrough (NDF) when all its outputs depend only on its state vector: $\boldsymbol{Y}^{[b]} = \boldsymbol{g}(\boldsymbol{X}^{[b]}(t_n))$. It is said Direct Feedthrough (DF) if at least one of its outputs is a direct function of the inputs: $\boldsymbol{Y}^{[b]} = \boldsymbol{g}(\boldsymbol{X}^{[b]}(t_n), \boldsymbol{U}^{[b]}(t_n))$. The generation DF outputs is mainly caused by modeling artifacts as the reduction of the different equations from 3D to 0D and the addition of some inputs/outputs with the model splitting.

It is clear that the bound of the global error on outputs found in (9.8) stays valid for Non Direct Feedthrough models.

9.3 Data dependencies cycles

9.3.1 Model decomposition

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 figure 9.3).

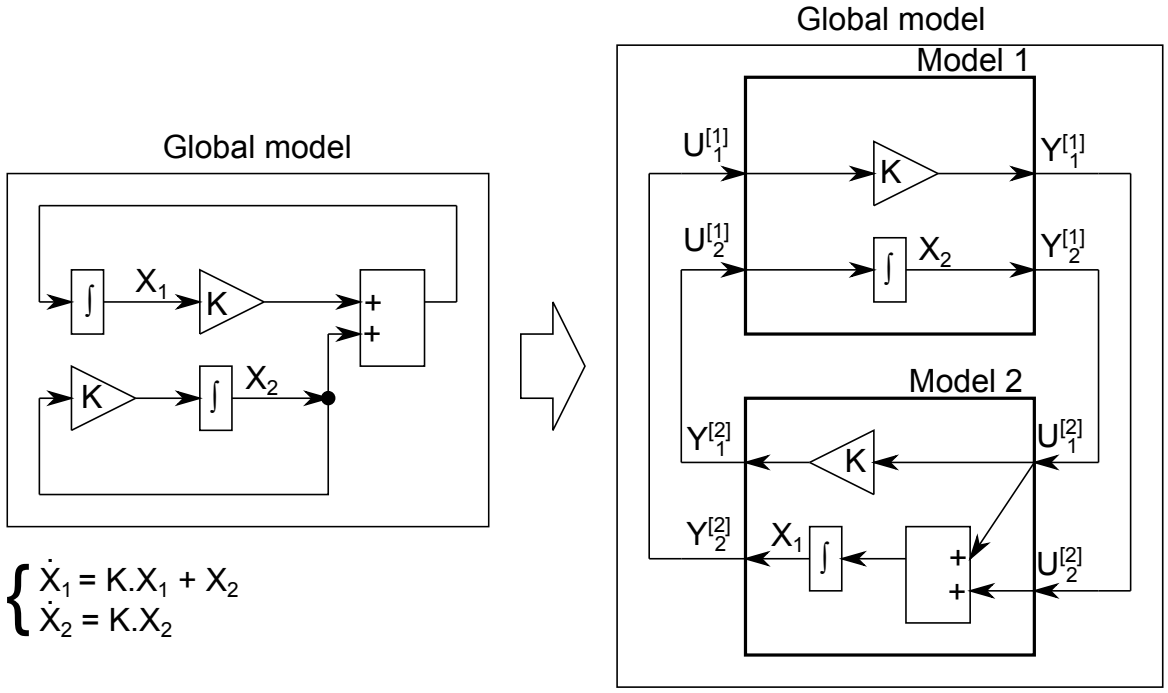

Figure 9.3: Loop creation due to the model splitting.

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, thus inducing simulation errors.

9.3.2 Model of computation with the modular co-simulation approach

As mentioned in section 9.3.1, breaking the created loop may lead to delayed outputs, depending on the models input/output properties. Two cases are considered.

1^{st} case: Coexistence of DF and NDF models

In this case, model 1 and model 2 are respectively considered NDF and DF. Since the initial conditions ($n = 0$) of $\boldsymbol{X}_n^{[1]}$ and $\boldsymbol{X}_n^{[2]}$ are known, only the outputs $\boldsymbol{Y}_n^{[1]}$ (and consequently $\boldsymbol{U}_n^{[2]}$) are ready to be computed. After their calculation, $\dot{\boldsymbol{X}}_n^{[2]}$ and $\boldsymbol{Y}_n^{[2]}$ (and consequently $\boldsymbol{U}_n^{[1]}$) are ready to run. Once $\boldsymbol{U}_n^{[1]}$ is available, the computing of $\dot{\boldsymbol{X}}_n^{[1]}$ is ready to be run. The same cycle is repeated until the end of simulation (see figure 9.4).

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

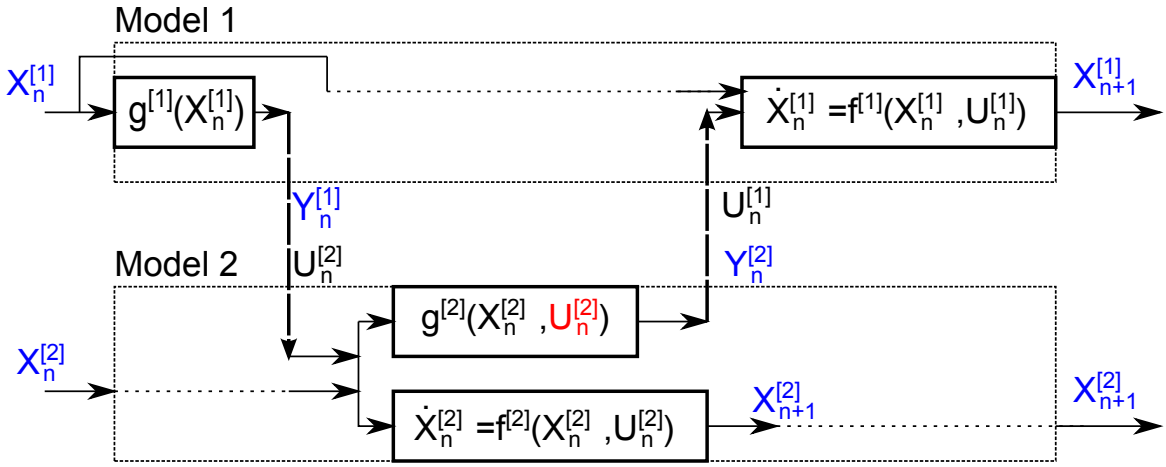

Figure 9.4: Defined execution order between NDF and DF models.

other words the whole (update outputs/update states) procedure takes place in a single instant ($t = t_n$) 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.

2^{nd} case: All the models are DF

In this case, model 1 and model 2 are both DF. At initial conditions ($n = 0$), $\boldsymbol{X}_n^{[1]}$ and $\boldsymbol{X}_n^{[2]}$ are known but none of $\boldsymbol{Y}_n^{[1]}$, $\boldsymbol{Y}_n^{[2]}$, $\dot{\boldsymbol{X}}_n^{[1]}$ and $\dot{\boldsymbol{X}}_n^{[2]}$ are ready to be computed. Indeed, $\boldsymbol{Y}_n^{[1]}$ and $\dot{\boldsymbol{X}}_n^{[1]}$ need $\boldsymbol{U}_n^{[1]} = \boldsymbol{Y}_n^{[2]}$, and at the same instant $\boldsymbol{Y}_n^{[2]}$ and $\dot{\boldsymbol{X}}_n^{[2]}$ need $\boldsymbol{U}_n^{[2]} = \boldsymbol{Y}_n^{[1]}$. This is a deadlock configuration and the loop is called algebraic. An execution order between model 1 and model 2 must be specified.

Regardless of the execution order, in our approach, it is inevitable to have at least a delayed model corresponding to the first executed one (see figure 9.5). 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.

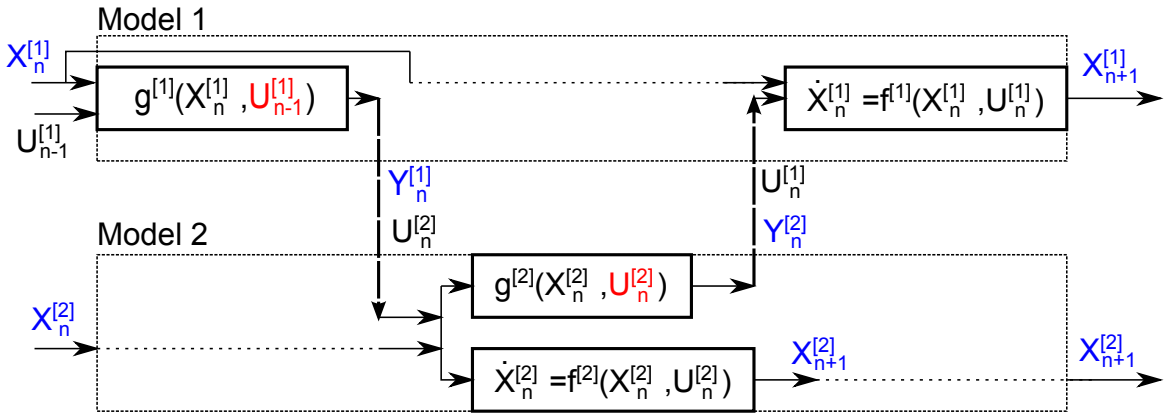

Figure 9.5: Delayed outputs due to the breaking of the algebraic loop.

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.

9.3.3 Error evaluation and convergence analysis for DF models with broken loops

The global error on the output of the first executed model $\boldsymbol{Y}^{[1]}$ is then defined as follow

$$\begin{aligned} \boldsymbol{Y}^{[1]}(t_n) - \tilde{\boldsymbol{Y}}_n^{[1]} &= \left(\boldsymbol{Y}^{[1]}(t_n) - \boldsymbol{Y}_n^{[1]}\right) + \left(\boldsymbol{Y}_n^{[1]} - \tilde{\boldsymbol{Y}}_n^{[1]}\right) \\ &= \boldsymbol{\Delta}_n(\boldsymbol{Y}^{[1]}) + \boldsymbol{\epsilon}_n(\boldsymbol{Y}^{[1]}) \end{aligned} \tag{9.9}$$

The global error on $\boldsymbol{Y}^{[1]}$ is caused by two terms. The first term, $\boldsymbol{\Delta}_n(\boldsymbol{Y}^{[1]})$, is due to the integration error (already bounded in (9.8)). The second term, $\boldsymbol{\epsilon}_n(\boldsymbol{Y}^{[1]})$, is due to the broken loop and its evaluation is performed as follow

$$\boldsymbol{\epsilon}_n(\boldsymbol{Y}^{[1]}) = \boldsymbol{Y}_n^{[1]} - \tilde{\boldsymbol{Y}}_n^{[1]} = \boldsymbol{g}^{[1]}(\boldsymbol{X}_n^{[1]}, \boldsymbol{U}_n^{[1]}) - \boldsymbol{g}^{[1]}(\boldsymbol{X}_n^{[1]}, \boldsymbol{U}_{n-1}^{[1]}). \tag{9.10}$$

Assuming now that $\boldsymbol{g}$ is Lipschitz continuous in $\boldsymbol{U}$ with $L > 0$ constant i.e.

$$\left\| \boldsymbol{g}^{[1]}(\boldsymbol{X}_n^{[1]}, \boldsymbol{U}_n^{[1]}) - \boldsymbol{g}^{[1]}(\boldsymbol{X}_n^{[1]}, \boldsymbol{U}_{n-1}^{[1]}) \right\| \leq L \left\| \boldsymbol{U}_n^{[1]} - \boldsymbol{U}_{n-1}^{[1]} \right\|,$$

then

$$\begin{aligned} \left\| \boldsymbol{Y}_n^{[1]} - \tilde{\boldsymbol{Y}}_n^{[1]} \right\| &\leq L \left\| \boldsymbol{U}_n^{[1]} - \boldsymbol{U}_{n-1}^{[1]} \right\| \\ &= \mathcal{O}(H)\,. \end{aligned} \tag{9.11}$$

Consequently

$$\left\| \boldsymbol{Y}^{[1]}(t_n) - \tilde{\boldsymbol{Y}}_n^{[1]} \right\| \leq \mathcal{O}(h^p) + \mathcal{O}(H)\,. \tag{9.12}$$

We can conclude that the global error on the outputs is clearly bounded by the numerical solver characteristics (time-step h and order p) and by the size of the communication step H. However, the communication step H dominates the error when it is chosen very wide compared to the integration step and plays an essential role in the trade-offs between integration speed and the results accuracy.

9.4 Error evaluation and convergence analysis for the parallel modular co-simulation

The case where all the models are run in parallel without respecting the execution order between the models represents the extended version of the modular co-simulation approach and denoted

"ev-MCosim". As it is shown in figure 8.3, all the data dependencies between the models are relaxed (broken), meaning that each model is executed using the last produced and saved values without waiting for the achievement the current computation.

9.4.1 Bound of the global error

The evaluation of the global error on outputs $\boldsymbol{Y}^{[1]}$ in (9.11) is generalized for $\boldsymbol{Y}^{[b]}$, with $b = 1, \ldots, B$.

For the global error on the states $\boldsymbol{X}^{[b]}$, with $b = 1, \ldots, B$, it is defined as follow

$$\begin{aligned} \boldsymbol{X}^{[b]}(t_{n+1}) - \tilde{\boldsymbol{X}}^{[b]}_{n+1} &= \left(\boldsymbol{X}^{[b]}(t_{n+1}) - \boldsymbol{X}^{[b]}_{n+1}\right) + \left(\boldsymbol{X}^{[b]}_{n+1} - \tilde{\boldsymbol{X}}^{[b]}_{n+1}\right) \\ &= \boldsymbol{\Delta}_{n+1} + \boldsymbol{\epsilon}_{n+1}. \end{aligned} \tag{9.13}$$

Similarly to $\boldsymbol{Y}^{[b]}$, the global error on $\boldsymbol{X}^{[b]}$ is the combination of the integration step $\boldsymbol{\Delta}_{n+1}$ and the error due to the broken loop $\boldsymbol{\epsilon}_{n+1}$.

The computation of

$$\boldsymbol{X}^{[b]}_{n+1} = \boldsymbol{X}^{[b]}_{n} + h\boldsymbol{\Phi}^{[b]}(\boldsymbol{X}^{[b]}_{n}, \boldsymbol{U}^{[b]}_{n}, \boldsymbol{f}^{[b]}), \tag{9.14}$$

is transformed to

$$\tilde{\boldsymbol{X}}^{[b]}_{n+1} = \tilde{\boldsymbol{X}}^{[b]}_{n} + h\boldsymbol{\Phi}^{[b]}(\tilde{\boldsymbol{X}}^{[b]}_{n}, \boldsymbol{U}^{[b]}_{n-1}, \boldsymbol{f}^{[b]}), \tag{9.15}$$

caused by the broken loop.

Subtraction (9.15) to (9.14), one can deduce

$$\begin{aligned} \boldsymbol{\epsilon}_{n+1} = \boldsymbol{X}^{[b]}_{n+1} - \tilde{\boldsymbol{X}}^{[b]}_{n+1} &= \boldsymbol{X}^{[b]}_{n} - \tilde{\boldsymbol{X}}^{[b]}_{n} \\ &+ h\left(\boldsymbol{\Phi}^{[b]}(\boldsymbol{X}^{[b]}_{n}, \boldsymbol{U}^{[b]}_{n}, \boldsymbol{f}^{[b]}) - \boldsymbol{\Phi}^{[b]}(\tilde{\boldsymbol{X}}^{[b]}_{n}, \boldsymbol{U}^{[b]}_{n}, \boldsymbol{f}^{[b]})\right) \\ &+ h\left(\boldsymbol{\Phi}^{[b]}(\tilde{\boldsymbol{X}}^{[b]}_{n}, \boldsymbol{U}^{[b]}_{n}, \boldsymbol{f}^{[b]}) - \boldsymbol{\Phi}^{[b]}(\tilde{\boldsymbol{X}}^{[b]}_{n}, \boldsymbol{U}^{[b]}_{n-1}, \boldsymbol{f}^{[b]})\right) \end{aligned} \tag{9.16}$$

Then, norms and the triangle inequality are applied to obtain

$$\begin{aligned} \left\|\boldsymbol{X}^{[b]}_{n+1} - \tilde{\boldsymbol{X}}^{[b]}_{n+1}\right\| &\leq (1+hL)\left\|\boldsymbol{X}^{[b]}_{n} - \tilde{\boldsymbol{X}}^{[b]}_{n}\right\| + hL\left\|\boldsymbol{U}^{[b]}_{n} - \boldsymbol{U}^{[b]}_{n-1}\right\| \\ &\leq \frac{(1+hL)^{(n+1)} - 1}{hL} hL \max_{0\leq k\leq n}\left\|\boldsymbol{U}^{[b]}_{k} - \boldsymbol{U}^{[b]}_{k-1}\right\| \\ &\leq (e^{t_{n+1}-t_0} - 1) \max_{0\leq k\leq n}\left\|\boldsymbol{U}^{[b]}_{k} - \boldsymbol{U}^{[b]}_{k-1}\right\| \\ &= \mathcal{O}(1).\mathcal{O}(H) \\ &= \mathcal{O}(H) \end{aligned} \tag{9.17}$$

Then, the error on the states with the "ev-MCosim" is

$$\left\| \boldsymbol{X}^{[b]}(t_{n+1}) - \tilde{\boldsymbol{X}}^{[b]}_{n+1} \right\| \leq \mathcal{O}(h^p) + \mathcal{O}(H)\,, \tag{9.18}$$

and then the conclusion is similar to "sv-MCosim", meaning that the error is directly related to the choice of the communication step-size and to the solver's nature. Besides, when the communication step H is taken very large compared to h, it dominates globally the simulation errors.

9.4.2 Parallelization induced integration errors

After analyzing the bound of the global error, using norms and inequalities, it is interesting to determine the global error expression to show the errors accumulation term.

To compute the next state value $\boldsymbol{X}^{[b]}(t_{n+1})$, with $b = 1,2$ (see figure 9.6), the numerical solver needs at least the values of $\boldsymbol{X}^{[b]}(t_n)$ and $\dot{\boldsymbol{X}}^{[b]}(t_n) = \boldsymbol{f}^{[b]}(\boldsymbol{X}^{[b]}(t_n), \boldsymbol{U}^{[b]}(t_n))$ (e.g. for Euler integration $\boldsymbol{\Phi}(t_n, \boldsymbol{X}_n, \boldsymbol{U}_n) = \boldsymbol{f}(t_n, \boldsymbol{X}_n, \boldsymbol{U}_n)$).

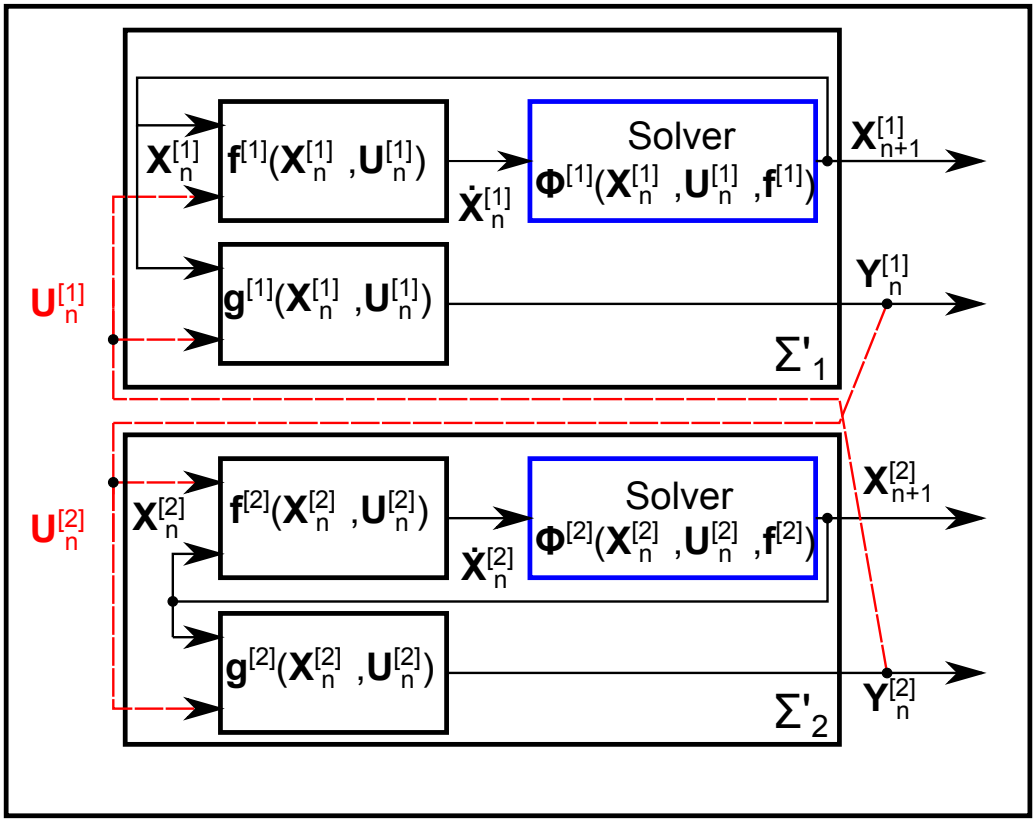

Figure 9.6: System's internal composition.

When computing $\dot{\boldsymbol{X}}^{[1]}_n = \boldsymbol{f}^{[1]}(\boldsymbol{X}^{[1]}_n, \boldsymbol{U}^{[1]}_n)$, the value of the local variable $\boldsymbol{X}^{[1]}_n$ is always available. This is not the case for $\boldsymbol{U}^{[1]}_n = \boldsymbol{Y}^{[2]}_n$, which is computed in a parallel branch. In fact, $\boldsymbol{Y}^{[2]}$ is only available in branch 1 at synchronization interval H, which is larger than the integration step h_n. In other words, $\boldsymbol{Y}^{[2]}_n$ is available only when the time t_n corresponds with a synchronization point t_s (see figure 9.2), otherwise its estimated value is the one transmitted at the previous

synchronization point. Let us evaluate the evolution of integration errors due to slack synchronization between the parallel branches when computing $\dot{\boldsymbol{X}}_n^{[1]} = \boldsymbol{f}^{[1]}(\boldsymbol{X}_n^{[1]}, \boldsymbol{Y}_n^{[2]})$. The analysis on Σ_1' remains valid for Σ_2'.

The influence of using a delayed value of $\boldsymbol{Y}^{[2]}$ in $\boldsymbol{f}^{[1]}(.)$ (respectively $\boldsymbol{Y}^{[1]}$ in $\boldsymbol{f}^{[2]}(.)$) is due to the lack of updated data during a delay τ, represented by the difference between the current integration time t_n and the last synchronization time t_s as

$$\tau = t_n - t_s, \tag{9.19}$$

with

$$t_s = H \left\lfloor \frac{t_n}{H} \right\rfloor,$$

therefore

$$t_s = \begin{cases} lH & \text{when} \quad t_n = l.H \quad l \in \mathbb{N}^*, \\ (l-1)H & \text{when} \quad t_n < l.H \quad l \in \mathbb{N}^*, \end{cases}$$

leading to

$$\begin{cases} \tau = 0 & \text{when} \quad t_n = t_s, \\ \tau > 0 & \text{when} \quad t_n > t_s. \end{cases}$$

To show up the delay τ, the expression of the induced error at t_{n+1} in the subsystem Σ_1', denoted $\boldsymbol{E}^{[1]}(t_{n+1})$, is written in the temporal form. It is the difference between $\boldsymbol{X}^{[1]}(t_{n+1})$ for the un-split model (9.20) and $\tilde{\boldsymbol{X}}^{[1]}(t_{n+1})$ for the split model (9.21):

$$\boldsymbol{X}^{[1]}(t_{k+1}) = \boldsymbol{X}^{[1]}(t_k) + h_k \boldsymbol{\Phi}^{[1]}(\boldsymbol{X}^{[1]}(t_k), \boldsymbol{Y}^{[2]}(t_k), h_n, p_n, \boldsymbol{f}^{[1]}), \quad k \in \{0, \ldots, n\} \tag{9.20}$$

$$\tilde{\boldsymbol{X}}^{[1]}(t_{k+1}) = \begin{cases} \boldsymbol{X}^{[1]}(t_{k+1}) & k = 0, \\ \tilde{\boldsymbol{X}}^{[1]}(t_k) + h_k \boldsymbol{\Phi}^{[1]}(\tilde{\boldsymbol{X}}^{[1]}(t_k), \tilde{\boldsymbol{Y}}^{[2]}(t_k - \tau,) h_n, p_n, \boldsymbol{f}^{[1]}) & k \geq 1. \end{cases} \tag{9.21}$$

In other words,

$$\begin{aligned} \boldsymbol{E}^{[1]}(t_{n+1}) &= \sum_{k=0}^{n} \boldsymbol{E}^{[1]}(t_k) \\ &+ h_n[\boldsymbol{\Phi}^{[1]}(\boldsymbol{X}^{[1]}(t_n), \boldsymbol{Y}^{[2]}(t_n), h_n, p_n, \boldsymbol{f}^{[1]}) - \boldsymbol{\Phi}^{[1]}(\tilde{\boldsymbol{X}}^{[1]}(t_n), \tilde{\boldsymbol{Y}}^{[2]}(t_n - \tau), h_n, p_n, \boldsymbol{f}^{[1]})] \\ &= \boldsymbol{E}_p^{[1]}(t_n) + \boldsymbol{E}_c^{[1]}(t_{n+1}), \end{aligned} \tag{9.22}$$

where

$$\begin{aligned} \boldsymbol{E}_c^{[1]}(t_{n+1}) &= h_n[\boldsymbol{\Phi}^{[1]}(\boldsymbol{X}^{[1]}(t_n), \boldsymbol{Y}^{[2]}(t_n)) - \boldsymbol{\Phi}^{[1]}(\tilde{\boldsymbol{X}}^{[1]}(t_n), \tilde{\boldsymbol{Y}}^{[2]}(t_n - \tau), h_n, p_n, \boldsymbol{f}^{[1]})] \\ \boldsymbol{E}_p^{[1]}(t_n) &= \sum_{k=0}^{n} \boldsymbol{E}^{[1]}(t_k). \end{aligned} \tag{9.23}$$

Here $\boldsymbol{E}_c^{[1]}(t_{n+1})$ is the current error generated at t_{n+1} whatever a synchronization or not. So, the global decoupling error $\boldsymbol{E}^{[1]}(t_{n+1})$ is the result of the accumulation of past errors $\boldsymbol{E}_p^{[1]}(t_n)$ and the current error $\boldsymbol{E}_c^{[1]}(t_{n+1})$.

As a conclusion, to achieve a correct result, two conditions must be met for the current (local) error and the global error:

- $|\boldsymbol{E}_c^{[1]}(t_{n+1})| < \boldsymbol{\epsilon}_{\text{loc}}$: allowed local error;
- $|\boldsymbol{E}^{[1]}(t_{n+1})| < \boldsymbol{\epsilon}_{\text{glo}}$: allowed global error.

These conditions can be satisfied by acting on some parameters. Indeed, in (9.23), the delay error depends on the integration steps h_n and on the delay τ. The integration step h_n is already adapted following the numerical solver strategy and the user-defined solver tolerance. The delay τ, however, depends on the last synchronization time t_s, which is function of the synchronization interval H.

The delay induced error tends to zero when the delay τ tends to zero, which means that the delay error can be eliminated with the synchronization interval set equal to the integration steps. In other words, all the parallel subsystems should be integrated at the same adaptive rate (in the case of adaptive synchronization period), or with same fixed time-step. These two assumptions are very restrictive, as they force to choose a global adequate time-step regardless the discontinuities and the stiffness of the subsystems. Compared with the single-threaded simulation, the only possible speed-ups during a parallel execution would be brought by the brute force computation power of the multi-core machine, reduced by the parallelization cost.

Therefore, considering a split model and a parallel execution, a trade-off must be found between acceptable simulation errors, thanks to tight enough synchronization, and simulation speed-ups thanks to decoupling between sub-models.

9.5 Generalization of the error bound for multi-rate co-simulation

Until now, error analyzes described in (9.5), (9.8), (9.12) and (9.18) were targeting the mono-rate modular co-simulation. The results showed that error on states and outputs are bounded by $\mathcal{O}(H)$ from the communication step point of view.

The aim of this section is to generalize these results to the multi-rate modular co-simulation case. This means that it is considered that each sub-model $\sum_b'$ (for $b = 1, \ldots, B$) can has its own communication rate $H^{[b]}$ different from the others, as it is shown in figure 9.7. A very simple idea is to say that the bound of these inequalities can be transformed to $\mathcal{O}(h^p) + \mathcal{O}\left(H^{[b]}\right)$.

However, this concept cannot be true since if we consider for example two connected blocks, where one of them has a rate x times faster of the other (see figure 9.7), the error on communication step is in reality bounded by $\mathcal{O}\left(\max_{b=1,\ldots,Bc} H^{[b]}\right)$ where B_c is the set of connected blocks. In the example of figure 9.7, the errors of $\sum_1'$ and $\sum_2'$ are bounded by $\mathcal{O}\left(H^{[2]}\right)$.

To conclude, the generalization of the previous results in (9.5), (9.8), (9.12) and (9.18) to the multi-rate case, is performed by replacing the term $\mathcal{O}(H)$ by the term $\mathcal{O}\left(\max_{b=1,\ldots,Bc} H^{[b]}\right)$, where B_c is the set of connected sub-models.

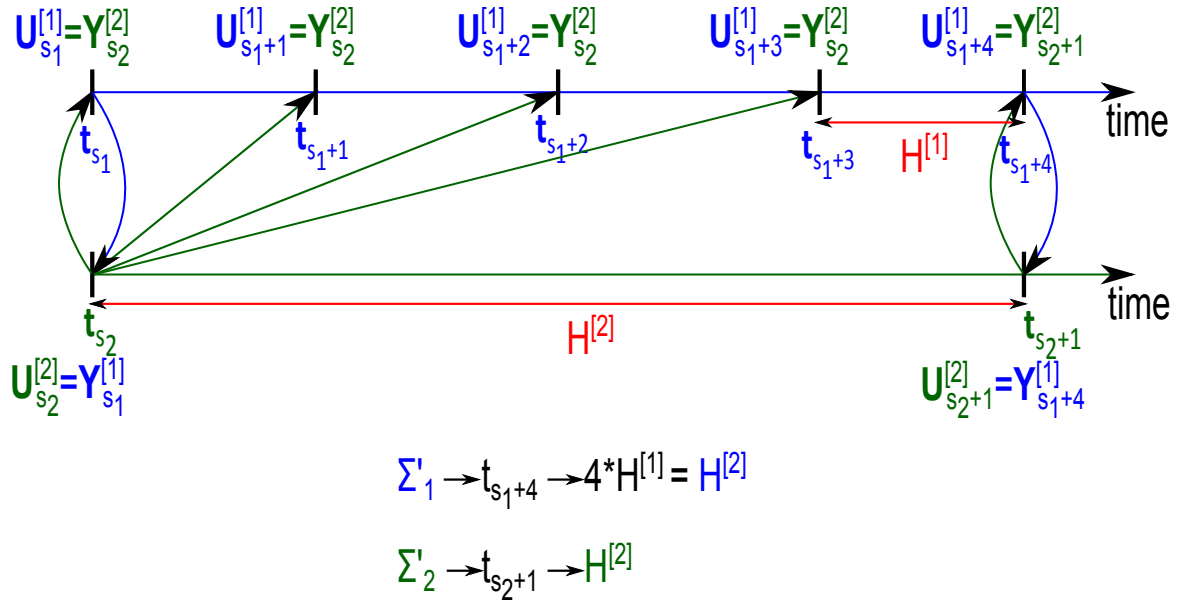

Figure 9.7: Multi-rate co-simulation.

9.6 Step-size control for communication intervals

To add a degree of freedom to this trade-off achievement, we propose to evaluate the induced error due to the communication step. The proposed approach is based on a similar work with numerical solver (see figure 9.8) and it is intended for mono-rate co-simulations (a common communication step).

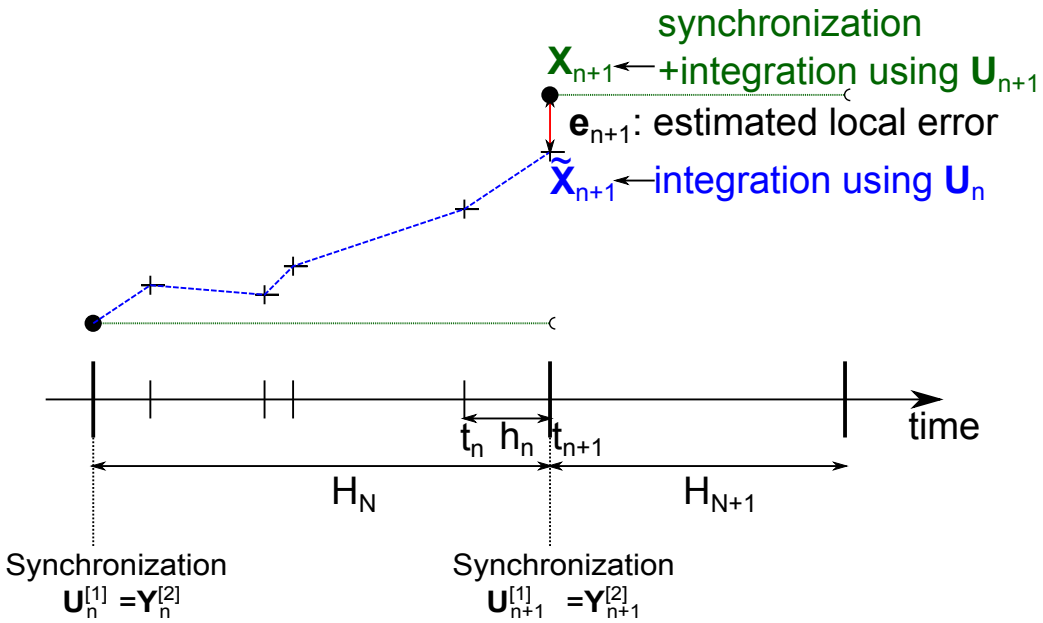

Figure 9.8: Estimated local error for communication step-size control.

This kind of step-size control requires the ability for states' roll-back, which means that the states of each sub-model must be saved each communication step.

The error indicator $E^{[b]}$,$(i = 1, \ldots, B))$ is computed such that

$$E_{n+1}^{[b]} = \sqrt{\frac{1}{n_{X^{[b]}}} \sum_{i=1}^{n_{X^{[b]}}} \left(\frac{e_{i,n+1}^{[b]}}{\mathrm{RTol}_i \left| X_{i,n+1}^{[b]} \right| + \mathrm{ATol}_i} \right)^2}. \tag{9.24}$$

Where $\boldsymbol{e}_{n+1}$ is the vector of estimated local errors $e_{i,n+1}$, computed at t_{n+1}. It is the difference between the state values computed with the solver and based on $\boldsymbol{U}_n$ and the state values computed with the current value of $\boldsymbol{U}_{n+1}$.

The solution $\boldsymbol{X}_{n+1}^{[b]}$ is accepted as sufficiently accurate if

$$E_{n+1}^{[b]} \leq 1 \tag{9.25}$$

is satisfied.

When at least one error indicator $E^{[b]}$ is greater than one, an ultimate indicator E_{n+1} is computed as follow

$$E_{n+1} = \max_{b=1,\ldots,B} E_{n+1}^{[b]},$$

and the next communication step-size is reduced

$$H_{n+1} = \alpha_s \frac{H_n}{E_{n+1}}, \tag{9.26}$$

with $\alpha_s \in [0.8, 0.9]$ is a safety factor. This reduction is performed until the condition in (9.25) is satisfied.

In the opposite case, to enlarge the next communication step following (9.26), all the models must satisfy (9.25).

This approach is a first proposal that is not tested yet. In fact, as it was mentioned earlier, the step-size control for the communication step requires the ability for states' roll-back. This ability was not provided with the FMI specification 1.0, which is currently implemented in the xMOD tool. It was only recently that it was provided with the FMI 2.0, released in october 2013.

9.7 Conclusion

This chapter presents the error evaluation and a convergence analysis for modular co-simulation. The bounding on the global error is performed for the states and the outputs and evaluated for different cases of the model of computation concerning the thread-level parallelism : "sv-MCosim", broken loops, "ev-MCosim". All the studies consider both mono-rate and multi-rate co-simulation and show that the error is tightly related to the coupling between the models, the numerical solver (order, time-step) and especially to the communication step. Finally, the last section proposes a method for adaptive communication step to limit the induced errors.

Chapter 10

Model decomposition based on structural analysis

10.1 Introduction

This chapter describes a methodology that can be used to split a system into several sub-models based on the block-diagonalization of incidence matrices that describes the different coupling and relationship between the state variables, the state derivatives and the events. The generated decomposition using this approach is independent from the physical meaning of the variables.

10.2 System splitting using block-diagonal forms

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 subsystems such that every discontinuity processing can be, as far as possible, encapsulated in a single subsystem.

The purpose is to optimize the exploitation of the parallelism of the subsystems while keeping under control the previously evaluated delay error in section 9.4.2 due the decoupling. Two methods have been analyzed for this aim, the first is related to the states to reduce the data-flow due to coupling variables between subsystems. The second one is related to the events, to reduce the number of integration interrupts, and also to minimize events detection and location via a complementary kind of parallelization through the solver.

10.2.1 Accounting for the state variables

To reduce the data exchange between two sub-models and to prioritize these exchanges inside one sub-model, the dependencies between the state variables must be evaluated. It can be done either by a direct access to the incidence matrix that describes the coupling between the state variables and their derivatives, or by computing the Jacobian matrix.

A Jacobian matrix is a matrix of all first-order partial derivatives of a vector function $\boldsymbol{f} = [f_1 f_2 \dots f_N]^T$ regarding another vector $\boldsymbol{X} = [x_1 x_2 \dots x_N]^T$.

An $N \times N$ Jacobian matrix denoted by J has the form:

$$\boldsymbol{J} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_N} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_N} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_N}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_N}{\partial x_N} \end{pmatrix}$$

If there is a zero element in the Jacobian, i.e. $\frac{\partial f_i}{\partial x_j} = 0$, it means that f_i is not influenced by x_j. However, f_i is actually $\dot{x}_i$. In other words, x_i does not depend on x_j. In the same way if $\frac{\partial f_i}{\partial x_j} \neq 0$, it means that x_i depends on x_j. Moreover, the numerical value of $\frac{\partial f_i}{\partial x_j}$ gives a measure of the sensitivity of $\dot{x}_i$ w.r.t. x_j.

This leads to conclude that the Jacobian matrix can be seen as an incidence matrix which provides useful information about data dependencies between state variables. This could be used for an effective system splitting. So that, when transforming the matrix into a block-diagonal form by permuting rows and columns, the blocks represent the independent subsystems. It may happen that a total block-diagonalization is not possible so that the final transformed matrix presents some coupling rows and/or column, this denotes the presence of irreducible dependencies between subsystems.

Note that this kind of use is only valid for linear systems because the Jacobian is constant. It can be however extended to nonlinear problems with invariant structure.

When the Jacobian matrix is not sparse enough to generate efficient blocks, instead of classifying the elements to only two categories: zero or non-zero, it would be interesting to take into account the weight of each element, by accessing the numerical values and defining some thresholds, to propose at the end some partial cuttings.

10.2.2 Accounting for the discontinuities

To minimize the delay error while optimizing the exploitation of the parallelism across the model, it is also crucial to reduce the number of discontinuities inside each sub-model, so that stiff variations of the state variables are limited. This procedure induces another benefit, as reducing the number of interrupts for each solver reduces re-starting overheads and improves the integration speed.

The events incidence matrix describes the relationships between events. Block-diagonalizing this matrix allows for separating the discontinuities and scatter them in the different sub-models.

Furthermore, the events incidence matrix block-diagonalization also leads to a kind of parallelization across the solver. In fact, the system resolution, including events handling, consists of 4 steps as mentioned in figure 10.1.

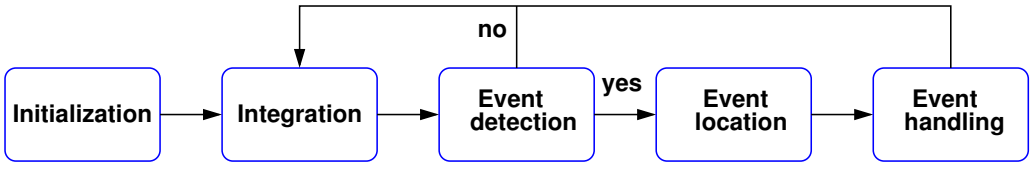

Figure 10.1: Events handling operations flow.

Event detection and location can be an expensive stage for hybrid systems (and for the addressed combustion models in particular). Especially, the event location (i.e. solving the zero-crossing equation (3.6)) can take a long time through an iterative process, and it is difficult to bound this step. By using the event incidence matrix, solving for a particular event can be localized in a subset of the global system through parallelization, thus shortening the zero-crossing function solving.

10.3 Permuting sparse rectangular matrices for block-diagonal forms

Two methods and associated software tools have been evaluated to perform the system diagonalization. Note that the original state variables of the system are preserved and that diagonal forms are produced only through permutations.

10.3.1 Bipartite graph model

A matrix $\boldsymbol{A}$ is transformed to a bipartite graph model. This graph is used by a specific tool to partition it, then to get a doubly bordered block-diagonal matrix $\boldsymbol{A}_{DB}$, i.e. the matrix has a block-diagonal form with non-zero elements on its last rows and columns as in figure 10.2.

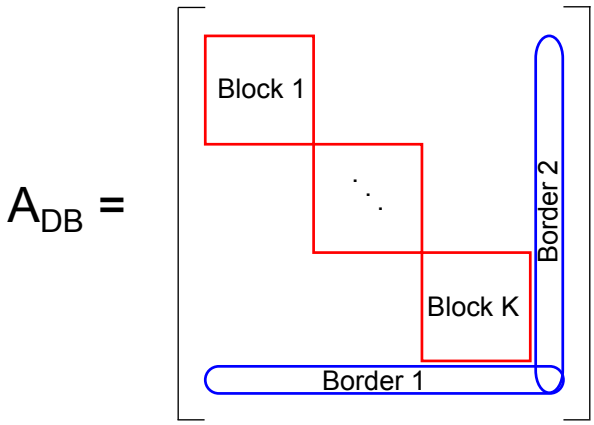

Figure 10.2: Doubly bordered block-diagonal matrix.

MeTiS [106] is a software aimed to partition large graphs. The used algorithms are based on multi-level graph partitioning, which means reducing the size of the graph by collapsing vertices and edges, then partitioning the smaller graph, and finally un-coarsening it to construct a partition for the original graph.

The block-diagonal form is performed by permuting rows and columns of a sparse matrix $\boldsymbol{A}$ to transform it into a K-way Doubly Bordered block-diagonal (DB) form $\boldsymbol{A}_{DB}$, where K is the number of blocks. It has a coupling row and a coupling column.

The representation of the non-zero structure of a matrix by a bipartite graph model reduces the permutation problem to those of Graph Partitioning by Vertex Separator (GPVS).

For example, let $\boldsymbol{A}$ the following matrix:

$$\boldsymbol{A} = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{10.1}$$

An undirected graph $G = (V, E)$ is defined as a set of vertices V and a set of edges E. The corresponding bipartite graph for MeTiS is built by replacing the rows and the columns by vertices and the non-zeros are represented by edges. After transformation, MeTiS partitions the graph as shown in figure 10.3.

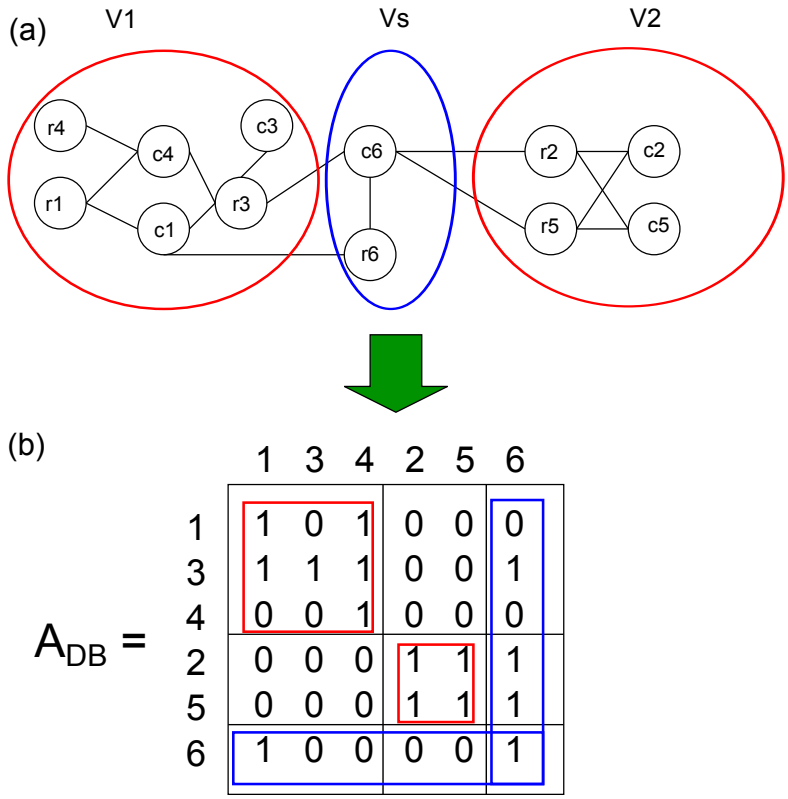

Figure 10.3: (a) Bipartite graph representation of the matrix $\boldsymbol{A}$ and 2-way partitioning of it by vertex separator Vs; (b) 2-way DB form of $\boldsymbol{A}$ induced by (a).

The objective of MeTiS when partitioning is to:

- Minimize the size of the separator because it implies the minimization of the border size;
- Balance among sub-bipartite graphs because it implies a balance among diagonal sub-matrices.

10.3.2 Hypergraph model

A matrix $\boldsymbol{A}$ is transformed to a hypergraph model. A hypergraph $H = (U, N)$ is defined as a set of nodes (vertices) U and a set of nets (hyper-edges) N among those vertices. This hypergraph is used by a specific tool to partition it, then to get a singly bordered block-diagonal matrix

$\boldsymbol{A}_{SB}$ as in figure 10.4, where the matrix has a block-diagonal form with non-zero elements only on its last rows.

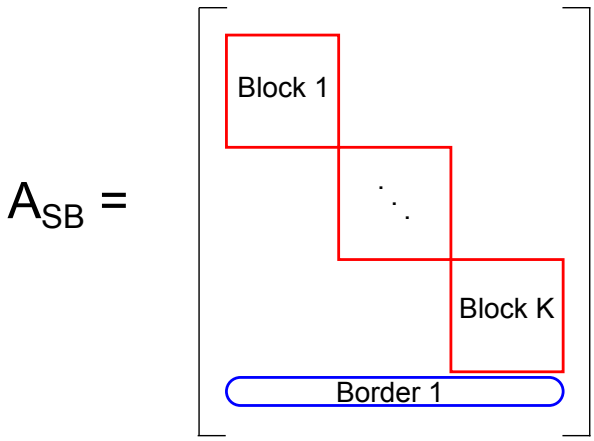

Figure 10.4: Singly bordered block-diagonal matrix.

PaToH (Partitioning Tools for Hypergraphs) [107] is a multi-level hypergraph partitioning tool that consist of 3 phases as for MeTiS: coarsening, initial partitioning, and uncoarsening. In the first phase, a multi-level clustering, that corresponds to coalescing highly interacting vertices to super-nodes, is applied on the original hypergraph by using different matching heuristics until the number of vertices drops below a predetermined threshold value. Then, the second phase corresponds to partition the coarsest hypergraph using diverse heuristics. Finally, in the third phase, the obtained partition is projected back to the original hypergraph by refining the projected partitions using different heuristics.

The block-diagonal form is performed by permuting rows and columns of a sparse matrix $\boldsymbol{A}$ in order to transform it into a K-way Singly Bordered block-diagonal (SB) form $\boldsymbol{A}_{SB}$. It has only a coupling row. For this reason, this method of block-diagonalization will be selected for the later study.

The representation of the non-zero structure of a matrix by a hypergraph model reduces the permutation problem to those of Hypergraph Partitioning (HP).

The corresponding hypergraph of the matrix $\boldsymbol{A}$ in (10.1) for PaToH is built by replacing the rows and the columns of the matrix by nets and nodes respectively. The number of pins is equal to the number of non-zeros in the matrix. After the transformation, PaToH partitions the hypergraph as it is shown in figure 10.5.

The objective of PaToH when partitioning is to:

- Minimize the cut size because it implies the minimization of the number of coupling rows;
- Balance among sub-hypergraphs because it implies a balance among diagonal sub-matrices.

In conclusion, the method using the bipartite graph model as MeTiS generates a doubly bordered block-diagonal matrix. To further reduce the coupling row and the coupling column to a single coupling row, the Ferris-Horn (FH) algorithm [108] uses a column splitting method. Unfortunately, the number of rows and columns of the matrix must be increased. In contrast, the method using the hypergraph model as PaToH directly generates a singly bordered block-diagonal matrix which means only a coupling row without adding an intermediate method. Therefore PaToH will be preferred to the block-diagonalization of matrices in the following case study.

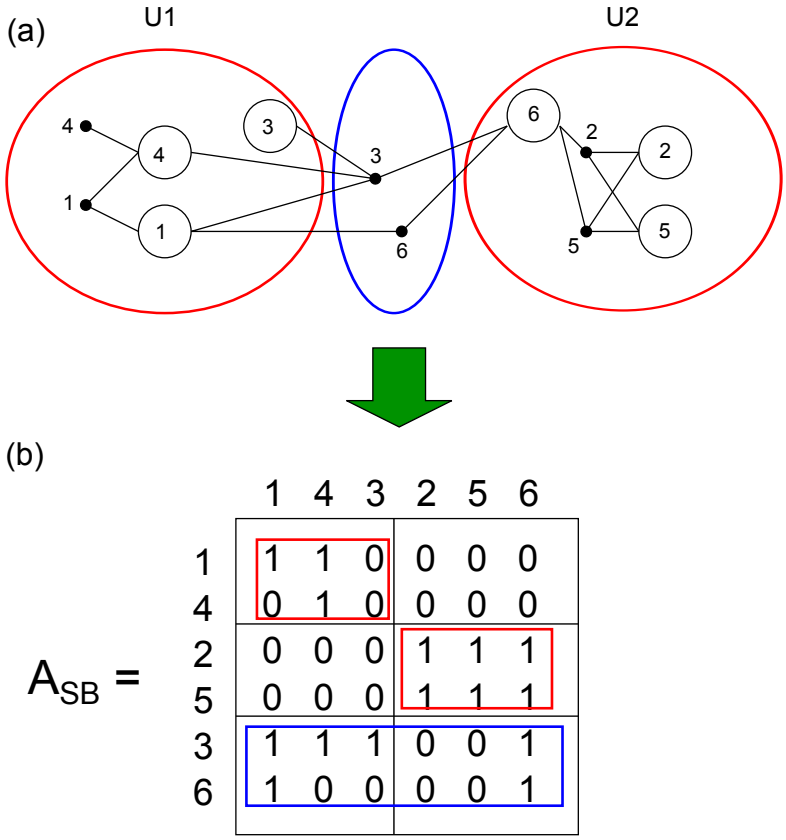

Figure 10.5: (a) Row-net hypergraph representation of the matrix $\boldsymbol{A}$ and 2-way partitioning of it; (b) 2-way SB form of $\boldsymbol{A}$ induced by (a).

10.4 Analysis of system splitting using a hypergraph model through a case-study

In the following tests, relationships between state variables and events as well as their behaviors are essential to study how to split the system at wisely chosen joints.

For this aim, the model, originally written in the Modelica language, was translated to a simpler language called Micro-Modelica (μ-Modelica) [109], which is understandable by the stand-alone Quantized State Systems (QSS) tool [56] as shown in figure 10.6.

Due to the lack of automated tool, this translation was made by hand. Adding to this limitation, the complexity and the size of the four-cylinder engine model, the study was restricted to a mono-cylinder engine model. Even with this model restriction, the translation was very time consuming to perform, including a thousandth of coding lines and referring sometimes to intuitions and hints for the debugging (the tool was not yet mature).

The QSS solver is not used here, only a related part of the tool-chain is used to generate a so-called simulation file which contains important information about the system and relationships between states and events. These data are extracted thereafter by a custom dedicated tool, and translated both to a matrix form for visualization and to a hypergraph file for the PaToH tool. Finally PaToH generates a partitioned hypergraph file that describes how the graph is decomposed and transformed subsequently to a matrix form for visualization.

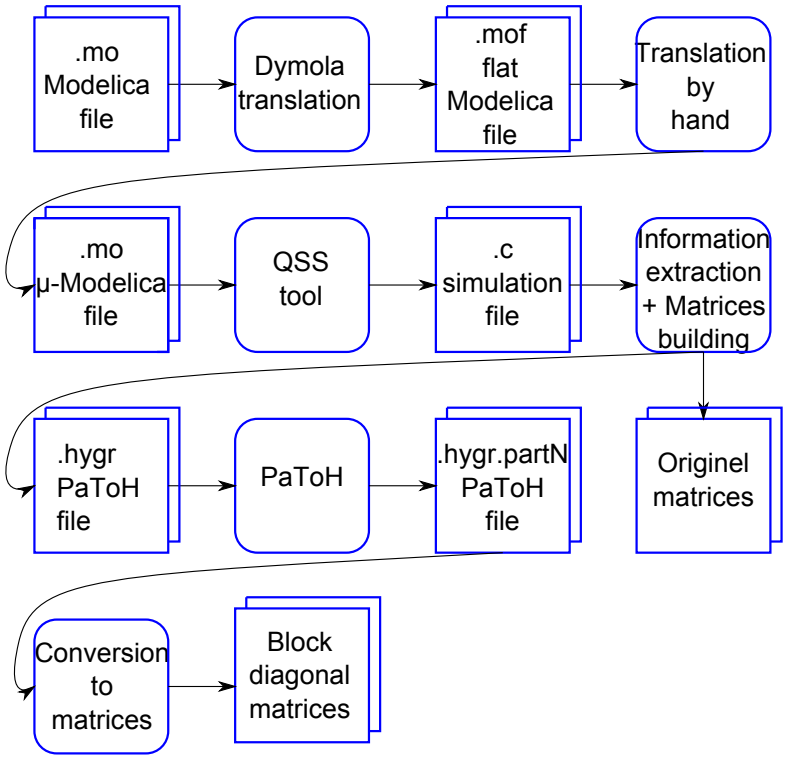

Figure 10.6: Software tool-chain.

The considered mono-cylinder model is characterized by a number of:

- State variables: $n_X = 15$;
- Events: $n_Z = 111$;
- Discrete variables: $n_D = 93$.

The state variables x_i $(i = 1, \ldots, n_X)$ are defined in table 10.1.

Table 10.1: State variables list.

ID	Name	Details
X_0	CrankAngle	Crank Shaft angle
$X_{1\to3}$	mEvapo[3]	Gas mass evaporated due to injection in global Mass balance equation
X_4	qvAlfa	Current released heat generated by the combustion process
X_5	mrefAlfa	Burned mass fraction during combustion process
X_6	combuHeatRelease	Output current combustion heat released
$X_{7\to9}$	mCombu[3]	Gas mass derivatives due to combustion in global Mass balance equation
$X_{10\to12}$	M[3]	Mass of gas
X_{13}	Energy	Energy contained in the cylinder
X_{14}	cylinderTemp	Output temperature in the cylinder

The events z_i $(i = 1, \ldots, n_Z)$ and discrete variables d_i $(i = 1, \ldots, n_D)$ are defined by the "when" blocks as follow:

```
when (z_i) then
 d_i = ... ;
elsewhen !(z_i) then
 d_i = ... ;
end when;
```

The statements that are between the "then" and the "elsewhen" or the "end when" are called the event handler, it represents the consequence of the event.

10.4.1 State and derivatives incidence matrices

At first glance, the number of coupled state variables is 6 among 15. In fact, $\dot{X}_{13}$ is only influenced by the state variables X_0, X_{10}, X_{11}, X_{12}, X_{14} as shown in figure 10.7.

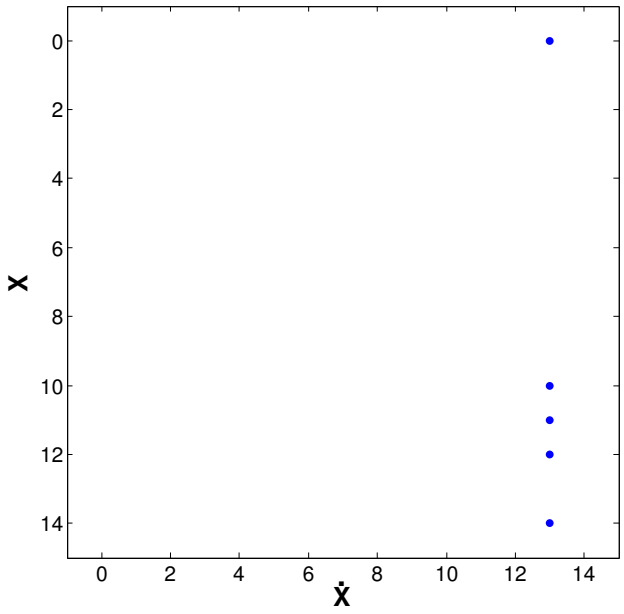

Figure 10.7: Incidence state matrix: derivatives of state variables $\dot{\boldsymbol{X}}$ depending on state variables $\boldsymbol{X}$.

Thus far, considering only the incidence state matrix, only 40% of the state variables are directly computed from the other states, while the others depend on external inputs (or even remain constant on some particular trajectories of the state space, e.g. when imposing a constant velocity of the crank).

The same result is found for events. In fact, the number of active events is 39 among 111, as the previous cited involved state variables directly affect values of 39 events as shown in figure 10.8.

This number represents only 35% of the total number of events, while the rest is only used to activate other events. In fact these 72 events are defined in the ModEngine library to be used in more general systems, not for the particular mono-cylinder use case. In consequence only the subset of active events must be detected.

However, if the state variables $\boldsymbol{X}$ can affect the events $\boldsymbol{Z}$, the events can also change the state variables values. In order to construct its corresponding matrix, both the incidence matrix that defines the discrete variables $\boldsymbol{D}$ influenced by the events $\boldsymbol{Z}$: $\boldsymbol{Z} \rightarrow \boldsymbol{D}$ (see figure 10.9) and the

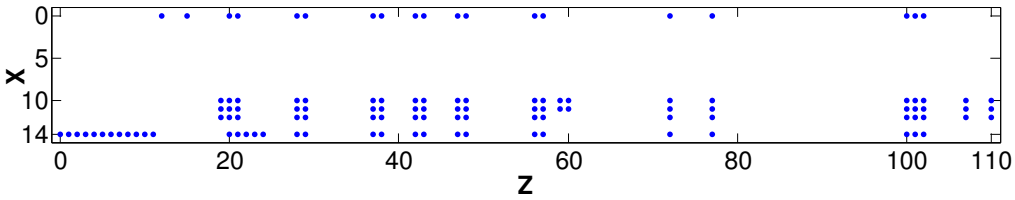

Figure 10.8: Incidence matrix: events $\boldsymbol{Z}$ depending on the state variables $\boldsymbol{X}$.

incidence matrix that defines the derivatives of the state variables $\dot{\boldsymbol{X}}$ influenced by the discrete variables $\boldsymbol{D}$: $\boldsymbol{D} \rightarrow \dot{\boldsymbol{X}}$ (see figure 10.10) are carried out.

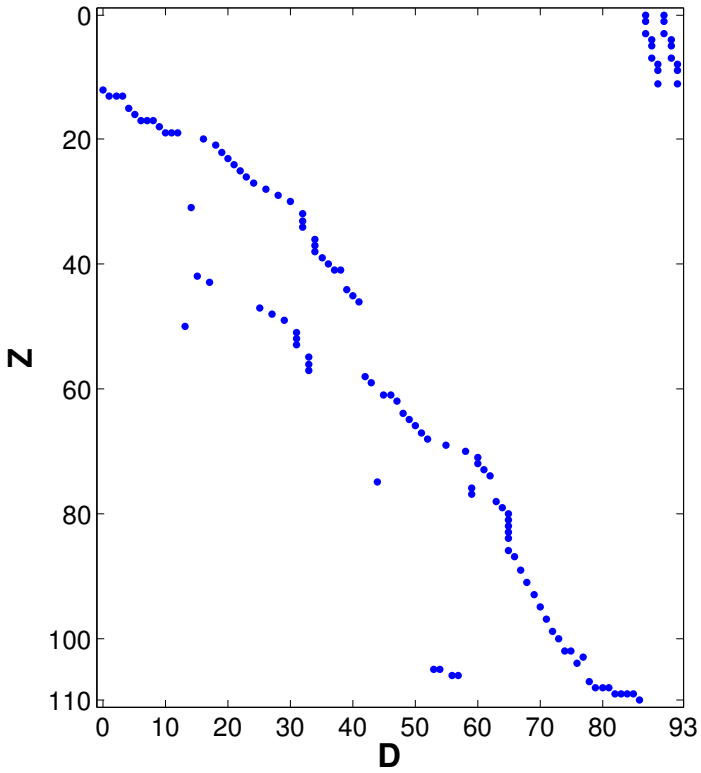

Figure 10.9: Incidence matrix: discrete variables $\boldsymbol{D}$ influenced by the events $\boldsymbol{Z}$.

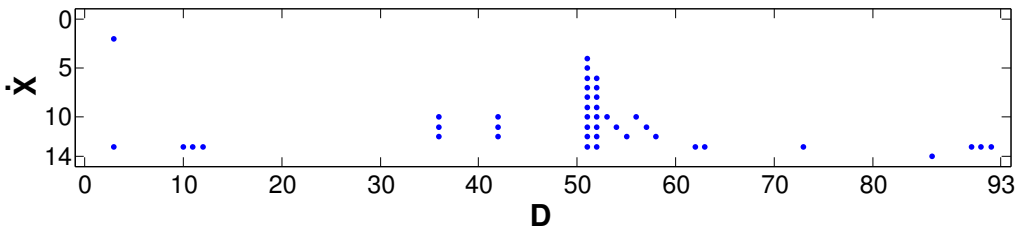

Figure 10.10: Incidence matrix: derivatives of state variables $\dot{\boldsymbol{X}}$ influenced by discrete variables $\boldsymbol{D}$.

Thus the incidence matrix $\boldsymbol{Z} \rightarrow \dot{\boldsymbol{X}}$ is deduced by transitivity in figure 10.11.

Figure 10.11 shows that the previous identification of some state variables as not coupled (based

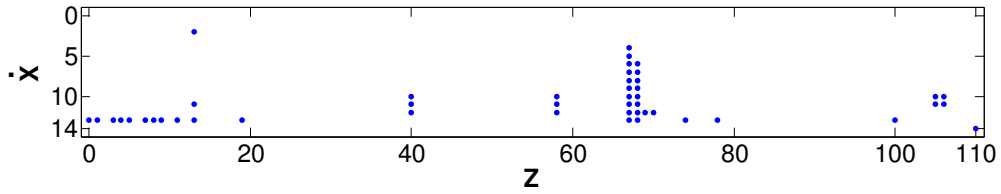

Figure 10.11: Incidence matrix: derivatives of state variables $\dot{\boldsymbol{X}}$ influenced by the events $\boldsymbol{Z}$.

on incidence state matrix figure 10.7) and events influenced by state variables (figure 10.8), is no longer true. In fact, now 13 state variables among 15 appear in this incidence matrix. Note that now only the state variables corresponding to X_1 and X_3 do not appear in this incidence matrix, this is due to the fact that these variables are inhibited momentarily to test a particular scenario.

By combining the two matrices in figures 10.8 and 10.11, an incidence matrix between events and state variables can be achieved as in figure 10.12.

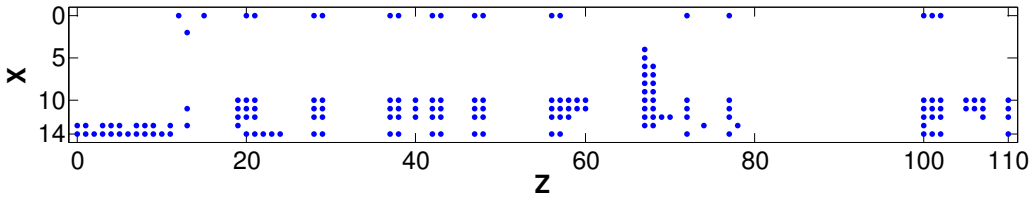

Figure 10.12: Incidence matrix: data exchange between events $\boldsymbol{Z}$ and state variables $\boldsymbol{X}$.

Once unnecessary states and events are eliminated and only the involved ones are kept, the intrication between state variables and events in both directions shows that it is difficult to separate or split the system.

Besides, from figure 10.8 ($\boldsymbol{X} \rightarrow \boldsymbol{Z}$) and figure 10.11 ($\boldsymbol{Z} \rightarrow \dot{\boldsymbol{X}}$), the state incidence matrix can be built differently than in figure 10.7, by passing through the events as it is shown in figure 10.13.

Using this construction through the events $\boldsymbol{Z}$, it appears that the state derivative $\dot{X}_{14}$ is also depending on X_{10}, X_{11} and X_{12}. Therefore, in order to determine correctly the relationships between the variables, it is important to use all the available system data, directly and by transitivity.

Finally, the relationships between the states' derivatives and the states, then between the states and the events show that the modeling of the mono-cylinder uses sequential dependency between the variables (or equations). This makes difficult, even impossible the parallelization of the model's execution. This discovery asserts the statement of engine specialists to not separate the combustion chamber model from the crankshaft model for example, because they share the same information at the same time.

An alternative to this result is to analyze the relationship between events to study the possibility of the separation of some events into blocks at the solver's root-finding level. This decomposition will facilitate the events detection and location.

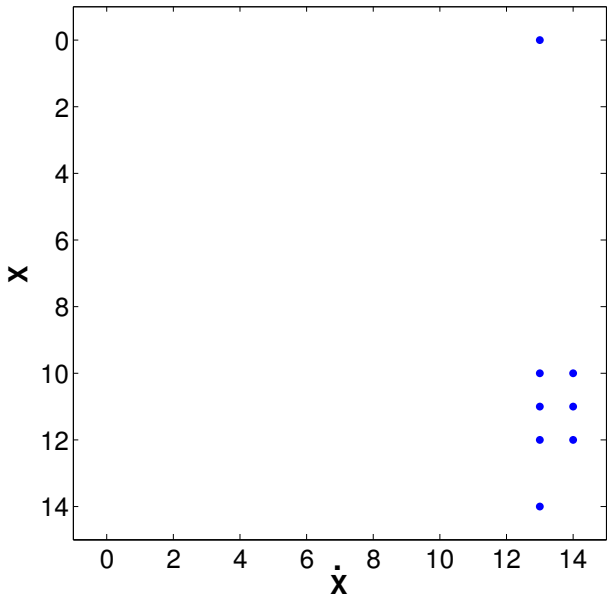

Figure 10.13: Incidence state matrix: derivatives of state variables $\dot{\boldsymbol{X}}$ influenced by state variables $\boldsymbol{X}$.

10.4.2 Incidence event matrix

The incidence event matrix can be built by transitivity. In fact, using the incidence matrix that define $\boldsymbol{Z} \rightarrow \boldsymbol{D}$ (see figure 10.9) and conversely the matrix that define $\boldsymbol{D} \rightarrow \boldsymbol{Z}$, the incidence event matrix $\boldsymbol{Z} \rightarrow \boldsymbol{Z}$ can be deduced as it is shown in figure 10.14.

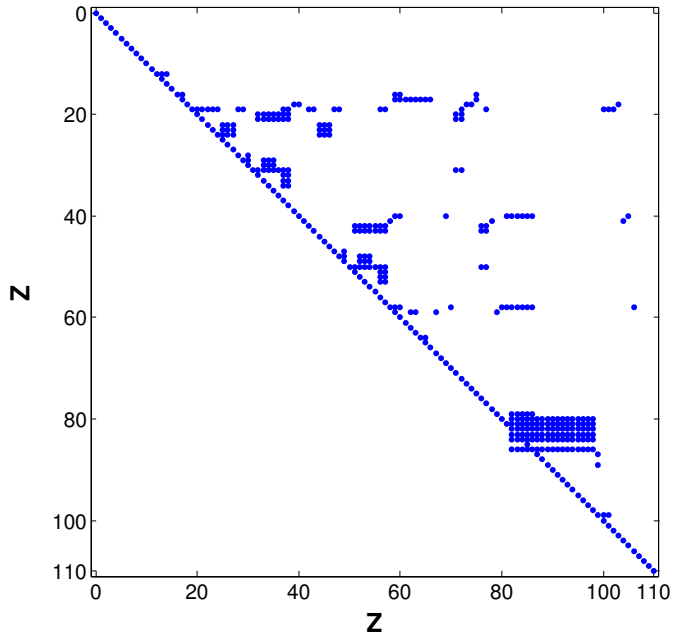

Figure 10.14: Incidence event matrix: events $\boldsymbol{Z}$ in columns influenced by events $\boldsymbol{Z}$ in rows.

As shown previously, it is hard to split the system based on the relationship between events $\boldsymbol{Z}$

and discrete variables $\boldsymbol{D}$. However, with the incidence event matrix, it is possible to transform it into a block-diagonal form with three blocks using PaToH and to consider each block as a subsystems where all the related discontinuities belong to the same entity (see figure 10.15).

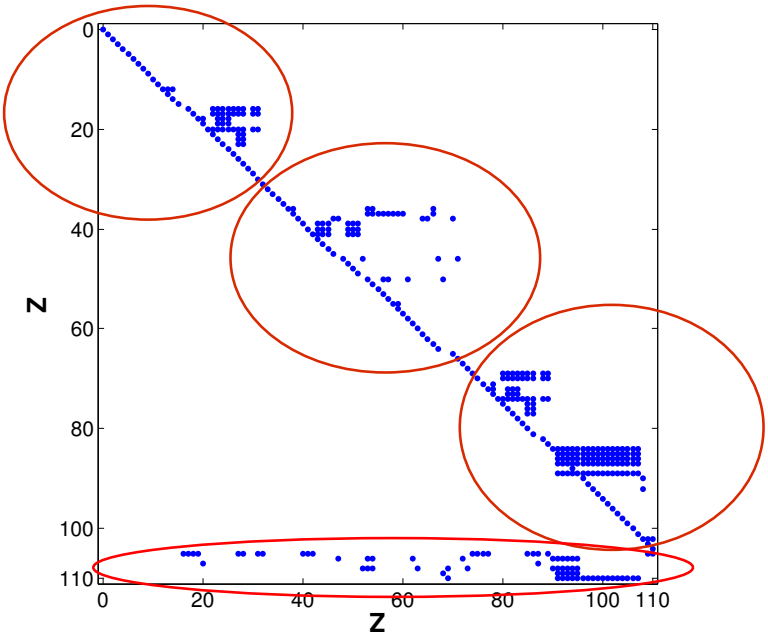

Figure 10.15: Block-diagonalized incidence event matrix: events $\boldsymbol{Z}$ in columns influenced by events $\boldsymbol{Z}$ in rows.

These blocks can be parallelized and we can hope the execution time to be reduced. In fact, the event detection, the event location and the restart of the solver increase the integration time as shown in figure 10.16 on page 99. In short, for the mono-cylinder integrated by the variable-step solver LSODAR, the average execution speed drops down to 4 times in case of events handling, and sometimes even up to 60 times. This confirms the interest on both limiting the number of interrupts inside each block of the model due to the events and parallelizing the event location through the solver. This expectation could not be experimentally tested due to the current unavailability of a runtime framework.

10.5 Conclusion

The particular case study shows that it is not always easy nor intuitive to know how to split a system, neither from a physical point of view nor from the relationship between the states and the events. In fact, when the matrix between the coupled states and events is not sparse, it is not possible to transform it into a block-diagonal form.

However, the incidence events matrix more likely seems to be sparse and its transformation to a block-diagonal form is feasible. Thus a relevant way to parallelize this particular system seems to perform it through the solver, leading to parallelize the steps corresponding to events handling which are costly for the numerical resolution (as already observed and plot in figure 10.16).

This chapter proposes some structural analysis techniques that may help the user to have an idea on how to partition a large system. The study analyzed a mono-cylinder model, which is an interesting case study since there is no obvious or intuitive partition. The results confirm

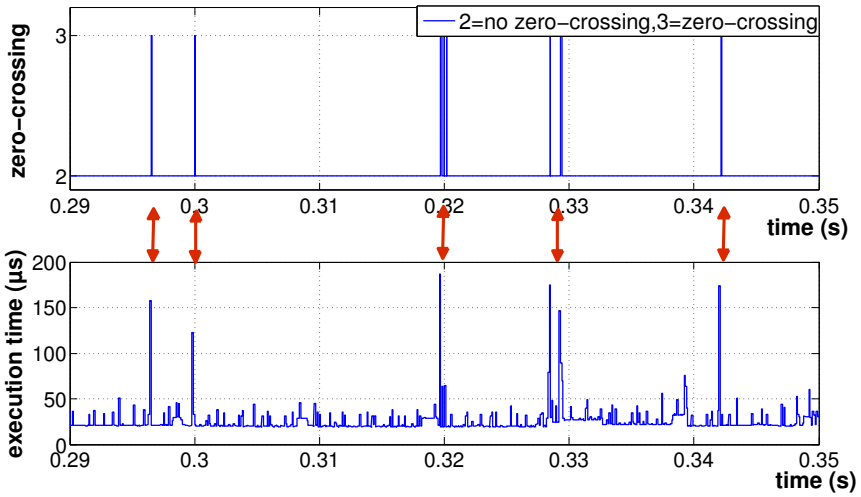

Figure 10.16: Effect of events handling on execution time.

that there is no universal automatic method for the splitting and that a minimum system's knowledge is required to do the job.

The four-cylinder model was nevertheless not analyzed since there is no provided automatic translator, needed for the generation of the dependencies matrices. It would be very interesting to be able to apply the decomposition method on such a model. In fact, the four-cylinder engine model was well-partitioned into 5 components (see chapter 8). This is efficient if we have at most 5 cores. However, when there are more available cores, the method presented in this chapter would be useful. Ultimately, since we have already a good case study of a partitioned model, all the proposed methods presented subsequently will be tested on this split four-cylinder engine model.

Finally, to practically evaluate the achievable speed-up, it is required to extend the tool-chain of figure 10.6, by developing a multi-thread runtime system able to take into account the parallelization choices. This development is out of the thesis scope since the current used model runtimes (FMUs) are not thread safe. HPC-OpenModelica project [110] aims to overcome this limitation and to allow multi-scale and thread-safe simulations. It is intended to bridge the gap between modern modeling tools and high-performance computing via standardization and implementation.

Chapter 11

Refined scheduling co-simulation

11.1 Introduction

Consider that the original complex model is already partitioned (following the previously described methods) into several lesser complex models. Even with an efficient execution order, when all the models are DF, delayed outputs still exist in the modular co-simulation approach (see section 9.3.2). To take advantage of the model splitting without adding useless delays, it deserves to look at the problem with a refined scheduling 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 (see figure 11.1).

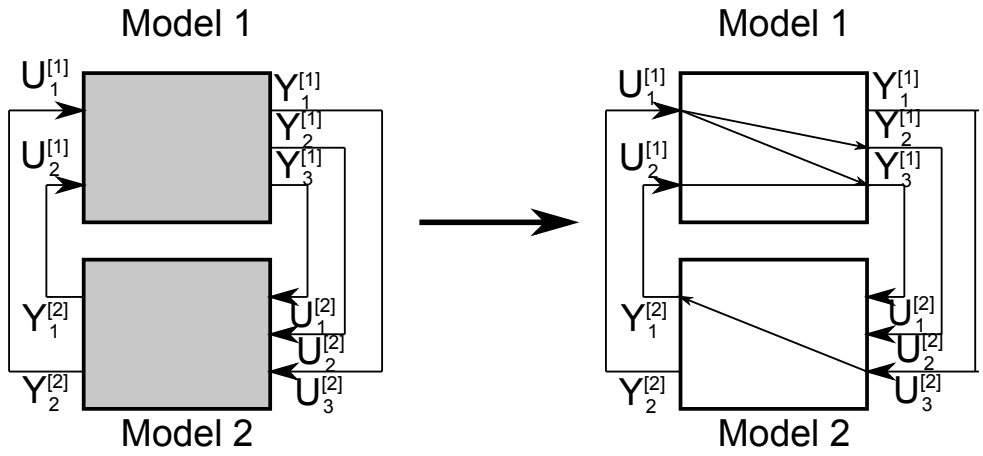

Figure 11.1: Input/Output connection through intra and inter models view.

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 figure 11.1, model 1 and model 2 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 3.5b), and a specific one to update the model states (i.e. integrate 3.5a). 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.

11.2 Refined dependency graph with the FMI specification

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 loops, such graph is necessarily a Direct Acyclic Graph (DAG) (see figure 11.2). 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 $Y_2^{[2]}$ to $U_1^{[1]}$ in figure 11.2). 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 $U_3^{[2]}$ to $Y_1^{[2]}$ in figure 11.2). 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 $U_2^{[1]}$ to $\dot{\boldsymbol{X}}^{[1]}$ in figure 11.2), 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 $Y_3^{[1]}$ to $\dot{\boldsymbol{X}}^{[1]}$ in figure 11.2), because the computation of $\boldsymbol{Y}_k$ needs $\boldsymbol{X}_k$ which is no longer available after $update_{\text{state}}$ computed $\boldsymbol{X}_{k+1}$. 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.

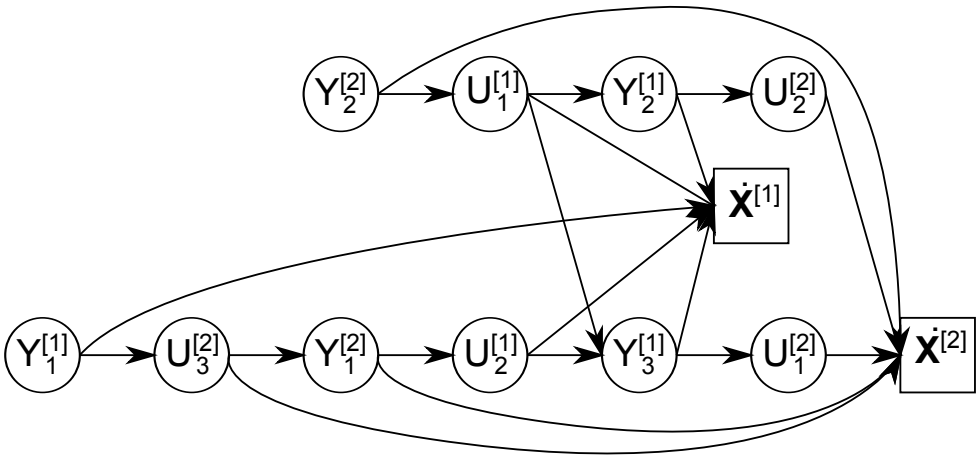

Figure 11.2: Refined dependency graph.

11.3 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 off-line heuristic approach (based on the algorithm architecture adequation methodology), similar to the one of [22], that tries to optimize the distribution and scheduling of the different model's operations (algorithm) on the different available cores (architecture).

The heuristic considers start dates and end dates for each operation and tends to minimize the critical path latency of a DAG, in which a computation time C_i is attached to each operation OP_i. For real-time simulation purpose, these computation times are estimated by

their WCET. Here, the goal is fast simulation, which is not safety critical, so that an estimated 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 $update_{\text{out}}$ (i.e. $C(update_{\text{state}}) \gg C(update_{\text{out}})$), while $update_{\text{in}}$ are simple data copy whose cost is negligible (i.e. $C(update_{\text{in}}) \ll \varepsilon$).

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.

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 for each operation OP_i and then the critical path CP as it is shown in Algorithm 2.

```
Initialization;
Set Ω the set of all the operations;
Set O the set of operations without predecessors;
foreach OP_i ∈ O do
    S_i := 0;
    E_i := S_i + C_i;
end
Set O' the set of operations whose all immediate predecessors were treated;
while O' ≠ ∅ do
    foreach OP_i ∈ O' do
        S_i := max(E_h : OP_h → OP_i), (OP_h are the immediate predecessors of OP_i);
        E_i := S_i + C_i;
    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
foreach OP_i ∈ Ω do
    if CP > E_i then
        CP := E_i;
    end
end
```

Algorithm 2: Computation of S_i, E_i and CP.

After that the computation of the start and end dates from the graph end denoted S_i^* and E_i^* and then the flexibility F_i can be performed for each operation OP_i as it is shown in Algorithm 3.

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

Initialization;
Set Ω the set of all the operations;
Set O the set of operations without successors;
foreach $OP_i \in$ O **do**
  $E_i^* := 0$;
  $S_i^* := E_i^* + C_i$;
end
Set O$'$ the set of operations whose all immediate successors were treated;
while O$' \neq \emptyset$ **do**
  foreach OP$_i \in$ O$'$ **do**
    $E_i^* := max(S_h^* : \text{OP}_i \rightarrow \text{OP}_h)$, (OP$_h$ are the immediate successors of OP$_i$);
    $S_i^* := E_i^* + C_i$;
    Remove OP$_i$ from the set O$'$;
    Add to the set O$'$ all predecessors of OP$_i$ for which all successors are already scheduled;
  end
end
foreach $OP_i \in \Omega$ **do**
  $F_i := \text{CP} - E_i - E_i^*$;
end

Algorithm 3: Computation of S_i^*, E_i^* and F_i.

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 $Y_1^{[1]}$ and $U_3^{[2]}$ are allocated by the heuristic on different cores, a semaphore is signalled just after $Y_1^{[1]}$ on its core, and a waiting semaphore operation is executed on the other core just before $U_3^{[2]}$.

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 4).

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 a powertrain case study.

11.4 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.

As mentioned in section 8.3, the split CFM-engine model (see chapter 7) gathers 91 inputs and 98 outputs. The scheduling of the refined co-simulation approach deals with 103 operations (5 $update_{\text{state}}$ and 98 $update_{\text{out}}$).

Initialization;
Set Ω the set of all the operations;
Set Γ the set of all the available cores;
foreach $\mathrm{OP}_i \in \Omega$ **do**
  Set $\mathrm{FixedCore}_i := \mathrm{NOT\_ALLOCATED}$; (operation OP_i is not already allocated);
end
foreach $\mathrm{Core}_j \in \Gamma$ **do**
  Set $T_{\mathrm{Core}_j} := 0$; (where T_{Core_j} corresponds to the first idle time on Core_j);
end
Set O the set of operations without predecessors;
while $\mathrm{O} \neq \emptyset$ **do**
  foreach $\mathrm{OP}_i \in \mathrm{O}$ **do**
    if $\mathrm{FixedCore}_i == \mathrm{NOT\_ALLOCATED}$ **then**
      Set cost_i to ∞; (cost of OP_i is set to the maximum value);
      foreach $\mathrm{Core}_j \in \Gamma$ **do**
        $S'_i := max(S_i, T_{\mathrm{Core}_j})$; (new start date of OP_i when executed on Core_j);
        $\mathrm{cost}_{i,j} := S'_i + C_i + E^*_i - \mathrm{CP}$; (cost of OP_i when executed on Core_j);
        if $\mathrm{cost}_{i,j} < \mathrm{cost}_i$ **then**
          Set $\mathrm{cost}_i := \mathrm{cost}_{i,j}$;
          Set $\mathrm{BestCore}_i := \mathrm{Core}_j$;
        end
      end
    else
      Set $\mathrm{BestCore}_i := \mathrm{FixedCore}_i$;
      $S'_i := max(S_i, T_{\mathrm{Core}_{\mathrm{BestCore}_i}})$;
      $\mathrm{cost}_i := S'_i + C_i + E^*_i - \mathrm{CP}$;
    end
  end
  Find OP_i with maximal cost_i in O;
  Schedule OP_i on its core $\mathrm{BestCore}_i$;
  Set $k := \mathrm{BestCore}_i$;
  $T_{\mathrm{Core}_k} := T_{\mathrm{Core}_k} + C_i$; (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**
      $\mathrm{FixedCore}_j := \mathrm{BestCore}_i$;
    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 4: Scheduling heuristic: minimization of cost function.

11.4.1 Model of computation

This study compares the simulation performance, observing the trade-offs between the simulation speed and simulation accuracy, for the two previous approaches of the Modular Co-simulation "MCosim" described in section 8.4 and the Refined Co-simulation denoted "RCosim" of the split model. For the case study, all the inputs and the outputs are updated following the order of scheduling heuristic, then the integration of the air path (AP) and the cylinders are performed in parallel.

The objective of the "RCosim" approach is to improve a little the results accuracy and reduce so much the simulation time compared to the "sv-MCosim" method. Moreover, compared to the "ev-MCosim" method, the aim is to improve so much the results accuracy at the cost of a potential little increase of the simulation time. The term "potential" is used because this increase may be balanced and even eliminated when using a solver with an error control.

Figure 11.3 reminds and summarizes the different methods. DT is the execution time during a communication step and it 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.

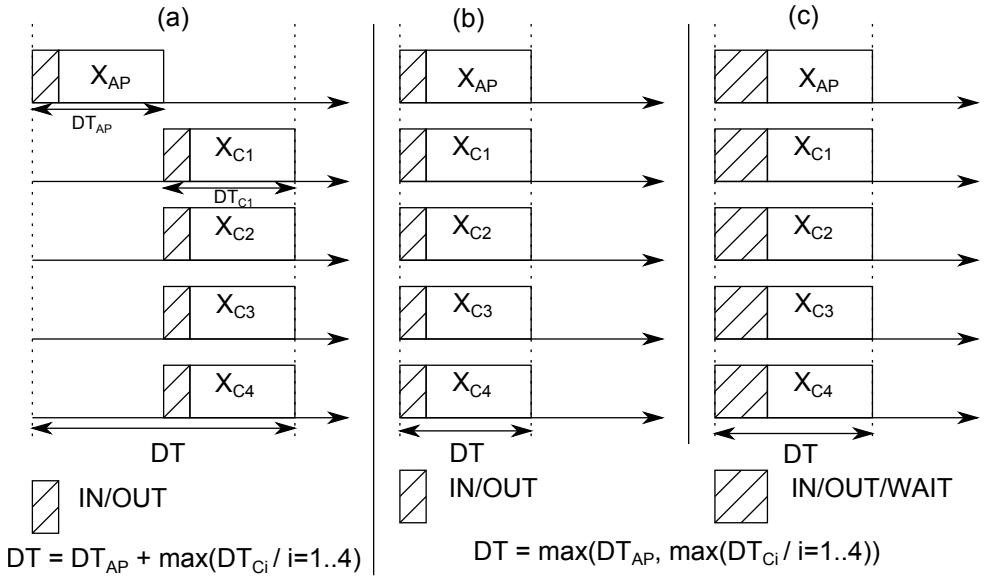

Figure 11.3: (a) sv-MCosim method; (b) ev-MCosim method; (c) RCosim method.

11.4.2 Reference simulations

The model validation is based on the observation of some quantities of interest as the intake and exhaust manifold pressures, air-fuel equivalence ratio and torque. These outputs are computed using LSODAR (see section 4.2.7).

The simulation state trajectory 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 reached for $\mathrm{Tol} = 10^{-7}$ (at the cost of an unacceptable slow simulation speed).

Then, to explore the trade-offs between the simulation speed and precision, simulations are run with increasing values of the solver tolerance until reaching a desired relative integration error Er, defined by (11.1)

$$Er(\%) = \frac{100}{N} \cdot \sum_{i=0}^{N-1} \left(\left| \frac{Y_{\text{ref}}(i) - Y(i)}{Y_{\text{ref}}(i)} \right| \right) \tag{11.1}$$

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

Iterations runs converge to a desired error ($Er \leq 1\,\%$) for $\text{Tol} = 10^{-4}$ (see table 11.1). The single thread simulation of the whole engine with LSODAR and $\text{Tol} = 10^{-4}$ provides the simulation execution time reference, to which the parallel versions are compared. When using the split model, each of its 5 components is assigned to a dedicated core and integrated by LSODAR with $\text{Tol} = 10^{-4}$.

Table 11.1: 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. Figure 11.4 illustrates the time-chart and shows the waiting period on the air path which it represents the difference between "sv-MCosim" and "ev-MCosim".

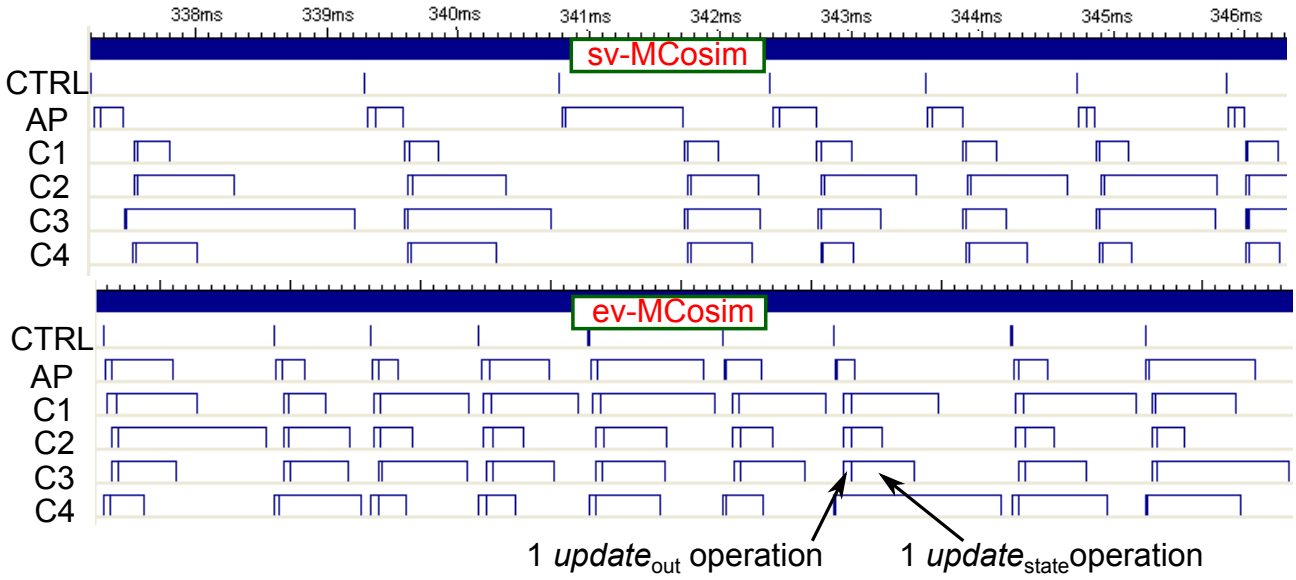

Figure 11.4: Modular co-simulation time-chart.

The refined co-simulation schedules the 103 $update_{\text{out}}$ and $update_{\text{state}}$ operations. As described in figure 11.5, 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.

11.4.3 Accuracy tests

DF outputs

The torque is a DF output of the air path, since it is the sum the four torques directly provided from each cylinder. Test results show that the torque is delayed by a communication step-size with the modular co-simulation method, as expected since all the models are DF. However,

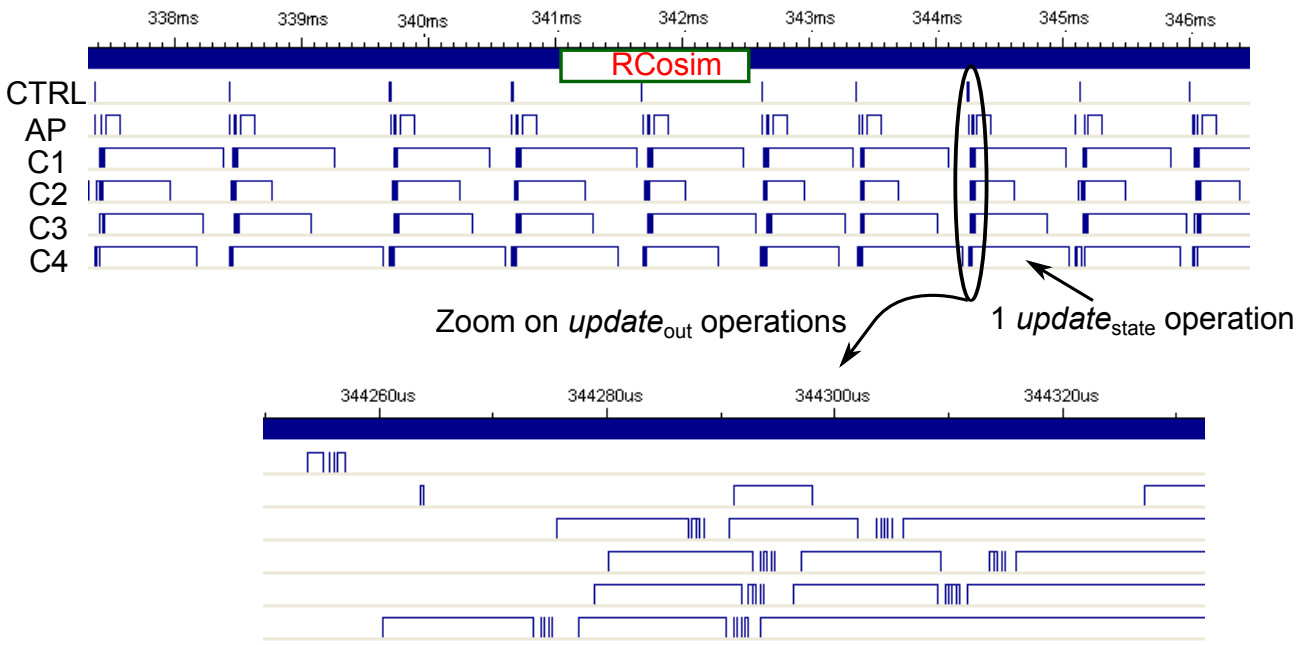

Figure 11.5: Scheduling of the update operations.

thanks to the refined co-simulation method, the torque is almost identical to the reference as indicated in figure 11.6.

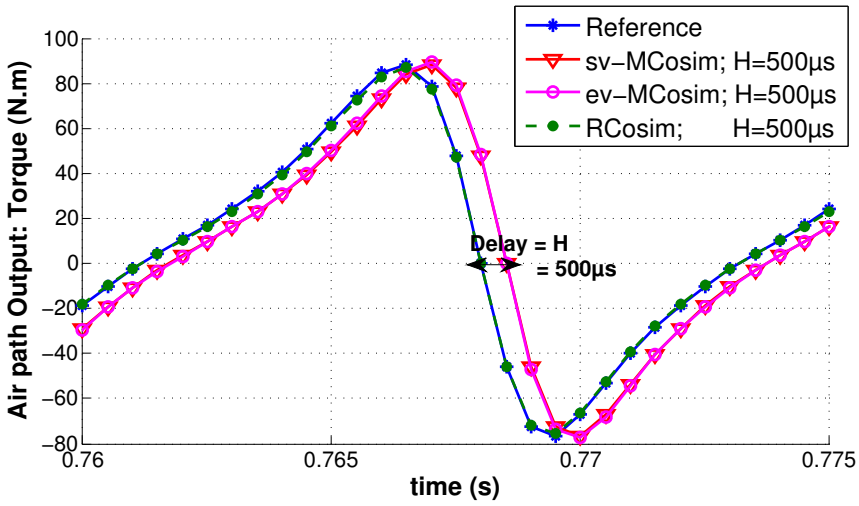

Figure 11.6: The behavior of a DF output regarding the different methods.

Then, the relative integration error is computed for several communication steps as in table 11.2. The results show that the refined co-simulation method keeps the integration stable even for large communication step. In fact, Er stays close to 1 %, whereas the modular co-simulation method suffers from delay-induced errors up to almost 20 %.

NDF outputs

The manifold pressure is a NDF output of the air path. For this case, there is no delay whatever the method.

Table 11.2: Relative integration error on the (DF) torque.

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

As for the torque, the relative integration error of the pressure also depends on the communication step (see figure 11.7). However, the step width is not so harmful as there are not loop-induced delays.

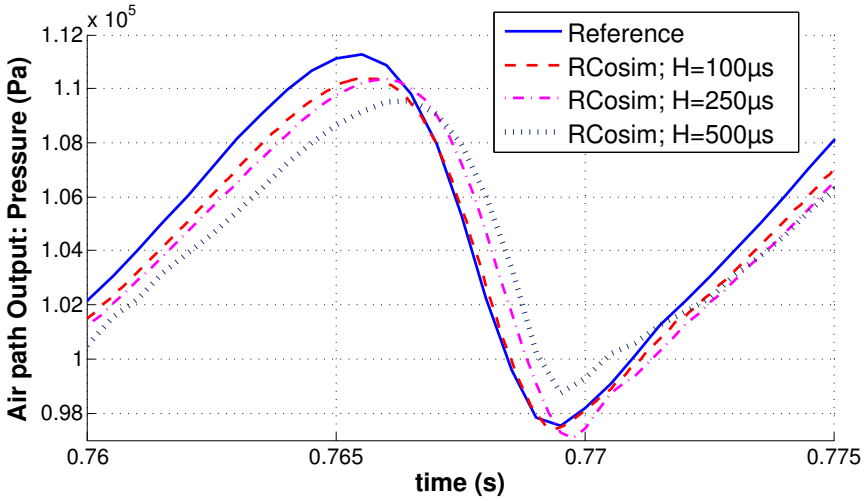

Figure 11.7: Effect of the communication step on a NDF output for RCosim method.

Table 11.3 shows that the refined method is again advantageous for the simulation accuracy. To reach the desired error both for DF and NDF outputs, the communication step must be restricted to 100 µs for modular co-simulation whereas it can be enlarged up to 500 µs with refined co-simulation.

Table 11.3: Relative integration error on the (NDF) manifold pressure.

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

11.4.4 Simulation time speed-up

The integration of the engine model (118 state variables and 312 event indicators) is time consuming. With $\text{Tol} = 10^{-4}$, 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 $H = 250\,\mu s$ (to keep $Er \approx 1\%$). table 11.4 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 variable step, the variable time-step solver rapidly finds the largest possible integration step to keep $Er \approx 1\,\%$. This later speed-up cannot be observed using fixed-step solvers.

Table 11.4: Simulation speed-up with the different approaches.

Simulation method	sv-MCosim	ev-MCosim	RCosim
Speed-up (5 cores)	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.

11.5 Conclusion

This chapter describes a new method of co-simulation that is based on a refined scheduling approach. The "RCosim" technique keeps the advantage of modular co-simulation that lies in the simulation speed-up. The speed-up is performed thanks to the parallel execution of the system's components. Besides, "RCosim" improves the accuracy of the simulation results through an off-line scheduling of operations that takes care of the models inputs/outputs dynamics. In conclusion, the combination of methods described in this chapter and the chapter 8 allows for supra-linear speed-ups of simulations on a multi-core architecture, while keeping the simulation precision under control.

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.

Chapter 12

Context-based extrapolation

12.1 Introduction

Considering a split model and a parallel execution, a trade-off must be found between acceptable simulation errors, thanks to tight enough synchronization, and simulation speed-ups thanks to decoupling between sub-models.

To add a degree of freedom to this trade-off achievement, we propose to extrapolate model inputs to compensate the stretching out of the communication steps between sub-models. In fact, it was proven in chapter 9 that the numerical solutions, in the modular co-simulation approach, are 1^{st} order accurate, $\mathcal{O}(H)$, when choosing larger communication step H. Considering the inputs held as constant between two synchronization intervals plays the role of a zeroth-order hold (constant extrapolation). To generalize the error bound, the $\mathcal{O}(H)$ term can be then replaced by $\mathcal{O}\left(H^{k+1}\right)$, where k is the extrapolation order. Using for example linear ($k = 1$) or quadratic ($k = 2$) extrapolation instead of constant ($k = 0$) extrapolation can then reduces the bound of simulation errors.

The difficulty in extrapolation is that it is sensitive for different reasons:

- prediction should be efficient: causal, sufficiently fast and reliable;
- there exist no universal prediction scheme, efficient for every signal;
- polynomial prediction may fail in stiff cases [42] (cf. Section 12.3 for details).

We choose to base our extrapolation on polynomial prediction, which allows fast and causal calculations. The rationale is that, in this situation, the computing cost of a low-order polynomial predictor would be by far smaller than the extra model computations needed by shorter communication steps. Since such predictions would be accurate neither for any signal (for instance, blocky versus smooth signals) nor any signal behavior (slow variations versus steep onsets), we borrow a context-based approach, common with loss-less image encoders [111], such as GIF or PNG formats. The general aim of these image coders is to predict a pixel value based on a pattern of causal neighboring pixels. Compression is obtained when the prediction residues possess smaller intensity values, and more generally a better distribution (concentrated around close-to-zero values) than the pixels in the original image. They may thus be coded on smaller "bytes", using entropy coding techniques. In images, one distinguish basic "objects" such as smooth-intensity varying regions, or edges with different orientations. Based on simple calculation of the prediction pattern pixels, different contexts are inferred (e.g. flat, smooth, $+45^o$ or -45^o edges, etc.). Look-up table predictors are then used, depending on the context.

In the proposed approach, we build a heuristic table of contexts (in Section 12.3) based on a short frame of past samples, and affect a pre-determined polynomial predictor to obtain a context-dependent extrapolated value. We now review the principles of extrapolation.

12.2 Causal polynomial prediction

12.2.1 Background on prediction

This section is dedicated to a peculiar instance of discrete time series or signal forecasting. The neighboring topics of prediction or extrapolation represent a large body of knowledge in signal processing [112], econometrics [113] or control [114].

In the present case, we consider a real-valued, regularly sampled signal u, with period P (that corresponds to the communication step H), known at synchronization or communication intervals. Prediction in general assumes the knowledge of signal formation models. Since very little is assumed on the signal's dynamics (no behavioral/explicit model is available, periodicity and regularity are unknown), and as we operate under real-time conditions, implying strong causality, only a tiny fraction of time series methods are practically applicable. Zeroth-order hold or nearest-neighbor extrapolation is probably the most natural, the less hypothetical, and the less computationally expensive forecasting method. It consists in using the latest known sample as the predicted value. It possesses small (cumulative) errors when the time series is relatively flat or its sampling rate is sufficiently high, with respect to the signal's dynamics. In other words, it is efficient when the time series is sampled fast enough to ensure small variation between two consecutive sampling times. However, it indirectly leads to under-sampling related disturbances, that affect the signal content. They appear as quantization-like noise, offset or peak flattening.

In our co-simulation framework, communication intervals are not chosen arbitrarily small for computational efficiency. Thus, the slow variation of inputs and outputs cannot be ensured in practice. Hence, borrowing additional samples from the past known data and using higher-order extrapolation methods could be beneficial, provided a trade-off of cost and error is met. Different forecast methods of various fidelity and complexity may be efficiently evaluated. We focus here on polynomial methods, for their simplicity and ease of implementation, following initial work in [1, Chapter 16].

12.2.2 Notations

We denote by $P_{(\delta,\lambda)}$ the least-squares polynomial predictor of degree $\delta \in \mathbb{N}$ and prediction length $\lambda \in \mathbb{N}^*$. The prediction length λ represents the number of past samples required for each prediction, performed in the least-squares sense [115, p. 227 sq.]. For convenience, we use a 0-last-sample-index convention: we re-index the frame of the λ past samples such that the last known sample is indexed by 0. Computations for the prediction at relative time τ (loosely denoted by $u(\tau)$), defined in (9.19), thus require samples $\{u_{1-\lambda}, u_{2-\lambda}, \ldots, u_0\}$. We first recall principles and formulas for a standard least-squares, degree-two or parabolic prediction. The general equations are derived next.

12.2.3 Polynomial prediction of degree $\delta = 2$

We look for the best fitting parabola, i.e. with degree $\delta = 2$, $u(t) = a_\delta + a_{\delta-1}t + a_{\delta-2}t^2$ to approximate the set of discrete samples $\{u_{1-\lambda}, u_{2-\lambda}, \ldots, u_0\}$. The prediction polynomial $P_{(2,\lambda)}$ is defined by the vector

$$\boldsymbol{a} = \begin{bmatrix} a_2 \\ a_1 \\ a_0 \end{bmatrix}$$

of polynomial coefficients. They are determined, in the least-squares sense [116], by minimizing the squared or quadratic, error:

$$e(\boldsymbol{a}) = \sum_{l=1-\lambda}^{0} \Big(u_l - (a_2 + a_1 l + a_0 l^2)\Big)^2 .$$

Note that the l indices here are non-positive, between $1-\lambda$ and 0. The minimum error is obtained by solving the following system of equations (zeroing the derivatives with respect to each of the free variables a_i):

$$\forall i \in \{0,1,2\}, \quad \frac{\partial e(\boldsymbol{a})}{\partial a_i} = 0$$

namely:

$$\begin{cases} \sum\limits_{l=1-\lambda}^{0} l^0 \Big(u_l - (a_2 l^0 + a_1 l^1 + a_0 l^2)\Big) = 0, \\ \sum\limits_{l=1-\lambda}^{0} l^1 \Big(u_l - (a_2 l^0 + a_1 i^l + a_0 l^2)\Big) = 0, \\ \sum\limits_{l=1-\lambda}^{0} l^2 \Big(u_l - (a_2 l^0 + a_1 l^1 + a_0 l^2)\Big) = 0. \end{cases} \tag{12.1}$$

The system in (12.1) may be rewritten as:

$$\begin{cases} \sum\limits_{1-\lambda}^{0} u_l = a_2 \sum\limits_{1-\lambda}^{0} l^0 + a_1 \sum\limits_{1-\lambda}^{0} l^1 + a_0 \sum\limits_{1-\lambda}^{0} l^2 , \\ \sum\limits_{1-\lambda}^{0} l u_l = a_2 \sum\limits_{1-\lambda}^{0} l^1 + a_1 \sum\limits_{1-\lambda}^{0} l^2 + a_0 \sum\limits_{1-\lambda}^{0} l^3 , \\ \sum\limits_{1-\lambda}^{0} l^2 u_l = a_2 \sum\limits_{1-\lambda}^{0} l^2 + a_1 \sum\limits_{1-\lambda}^{0} l^3 + a_0 \sum\limits_{1-\lambda}^{0} l^4 . \end{cases}$$

Let $m^d = \sum_{l=0}^{\lambda-1} l^{\delta-d} u_{-l}$ (here the indices l are positive) denote the $(\delta - d)$-th moment of the frame u_i, and $\boldsymbol{m}$ the vector of moments:

$$\boldsymbol{m} = \begin{bmatrix} m_2 \\ -m_1 \\ m_0 \end{bmatrix} .$$

We express the sums of integer powers by $\Sigma_\lambda^d = \sum_{i=0}^{\lambda-1} i^d$. Closed-form expressions exist for Σ_λ^d, involving Bernoulli sequences [117].

For instance:

- $\Sigma_\lambda^0 = \lambda$,
- $\Sigma_\lambda^1 = (\lambda-1)\lambda/2$,
- $\Sigma_\lambda^2 = (\lambda-1)\lambda(2\lambda-1)/6$,
- $\Sigma_\lambda^3 = (\lambda-1)^2\lambda^2/4$,
- $\Sigma_\lambda^4 = (\lambda-1)\lambda(2\lambda-1)(3\lambda^2-3\lambda-1)/30$.

We now form the matrix $\boldsymbol{Z}_{(2,\lambda)}$ of sums of powers (depending on $\delta = 2$ and λ):

$$\boldsymbol{Z}_{(2,\lambda)} = \begin{bmatrix} \Sigma_\lambda^0 & -\Sigma_\lambda^1 & \Sigma_\lambda^2 \\ -\Sigma_\lambda^1 & \Sigma_\lambda^2 & -\Sigma_\lambda^3 \\ \Sigma_\lambda^2 & -\Sigma_\lambda^3 & \Sigma_\lambda^4 \end{bmatrix}.$$

The system in (12.1) rewrites:

$$\begin{bmatrix} m^2 \\ -m^1 \\ m^0 \end{bmatrix} = \begin{bmatrix} \Sigma_\lambda^0 & -\Sigma_\lambda^1 & \Sigma_\lambda^2 \\ -\Sigma_\lambda^1 & \Sigma_\lambda^2 & -\Sigma_\lambda^3 \\ \Sigma_\lambda^2 & -\Sigma_\lambda^3 & \Sigma_\lambda^4 \end{bmatrix} \times \begin{bmatrix} a_2 \\ a_1 \\ a_0 \end{bmatrix}$$

or

$$\boldsymbol{m} = \boldsymbol{Z}_{(2,\lambda)} \times \boldsymbol{a}\,.$$

Now we want to find the value predicted by $P_{(2,\lambda)}$ at time τ. Let $\boldsymbol{\tau}_2$ be a vector of τ powers:

$$\boldsymbol{\tau}_2 = \begin{bmatrix} 1 \\ \tau \\ \tau^2 \end{bmatrix}.$$

Then, u_τ is equal to $a_2 + a_1\tau + a_0\tau^2 = \boldsymbol{\tau}_2^T \times \boldsymbol{a}$. Finally, $\boldsymbol{Z}_{(2,\lambda)}$ is always invertible, provided that $\lambda > \delta$. Its inverse is denoted $\boldsymbol{Z}_{(-2,\lambda)}$. It thus does not need to be updated in real-time. It may be computed off-line, numerically or even symbolically. Hence:

$$u(\tau) = \left(\boldsymbol{\tau}_2^T \times \boldsymbol{Z}_{(-2,\lambda)}\right) \times \boldsymbol{m}\,.$$

The vector $\boldsymbol{\tau}_2$ and $\boldsymbol{Z}_{(-2,\lambda)}$ are fixed, and the product $\boldsymbol{\tau}_2^T \times \boldsymbol{Z}_{(-2,\lambda)}$ may be stored at once. Thus, for each prediction, the only computations are the update of the vector $\boldsymbol{m}$ and its product with the aforementioned stored matrix. It thus enables look-up-table-based predictions, which helps to reduce propagation errors in matrix computations.

12.2.4 General formulas

Inferring from the previous example, we easily get a more generic extrapolation pattern in its matrix form.

$$u(\tau) = \begin{bmatrix} 1 & \tau & \cdots & \tau^\delta \end{bmatrix} \times \begin{bmatrix} \Sigma_\lambda^0 & -\Sigma_\lambda^1 & \cdots & (-1)^\delta \Sigma_\lambda^\delta \\ -\Sigma_\lambda^1 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ (-1)^\delta \Sigma_\lambda^\delta & \cdots & \cdots & \Sigma_\lambda^{2\delta} \end{bmatrix}^{-1} \times \begin{bmatrix} m^\delta \\ -m^{\delta-1} \\ \vdots \\ (-1)^\delta m^0 \end{bmatrix}.$$

Note

$$\boldsymbol{\tau}_\delta = \begin{bmatrix} 1 \\ \tau \\ \vdots \\ \tau^\delta \end{bmatrix},$$

then:

$$u(\tau) = \boldsymbol{\tau}_\delta^T \boldsymbol{Z}_{(-\delta,\lambda)} \boldsymbol{m}.$$

As in the previous case, only $\boldsymbol{m}$ and one matrix product need be computed in real-time. When $\delta = 0$, one easily sees that:

$$u(\tau) = \frac{m^0}{\Sigma_0^\lambda} = \frac{u_{1-\lambda} + \cdots + u_0}{\lambda},$$

that is, the running average of past frame values, reducing to the zeroth-order hold when $\lambda = 1$. Although the matrix formulation is convenient, actual computation does not require true matrix calculus, especially for small degrees δ. For instance, $P_{(1,3)}$ yields the simple estimator form

$$u(\tau) = \frac{\tau}{2}(u_0 - u_{-2}) + \frac{1}{6}(5u_0 + 2u_{-1} - u_{-2}).$$

Similarly, $P_{(2,5)}$ yield:

$$\begin{aligned} u(\tau) \quad = \tfrac{1}{10280} \quad & [\tau^2(2 * u_0 - u_{-1} - 2 * u_{-2} - u_{-3} + 2 * u_{-4}) \\ & +8\tau(258 * u_0 + 128 * u_{-1} - u_{-2} - 129 * u_{-3} - 256 * u_{-4}) \\ & +(6172 * u_0 + 4110 * u_{-1} + 2052 * u_{-2} - 2 * u_{-3} - 2052 * u_{-4})]. \end{aligned}$$

12.3 Context-based extrapolation

Actual complex systems usually present non-linearities and discontinuities, so that it is hard to predict their future behavior from past observations. Moreover the considered models are generated using the FMI for Model Exchange framework, which does not provide the inputs'

derivatives (conversely with the FMI for Co-Simulation architecture). Hence the previously described polynomial prediction cannot correctly extrapolate along all the system trajectories.

For example, [105] studies a method based on a sequential implementation of continuous dynamical systems that uses a constant, linear or quadratic extrapolation and a linear interpolation to improve the accuracy of the modular time integration. The study shows that the method is successful for non-stiff systems but it fails for the stiff case.

Our purpose is to define a method intended for the parallel simulation of hybrid dynamical systems. The context-based extrapolation is then performed to account for steps, stiffness, discontinuities or weird behavior, and use adapted extrapolation to limit excessively wrong prediction.

Keeping with the previous 0-last-sample-index convention, and for the sake of simplicity, we first define a measure of variation based on the last three samples: $d_0 = u_0 - u_{-1}$ and $d_1 = u_{-1} - u_{-2}$, the last and previous differences. Their absolute values are compared with two thresholds, γ_0 and γ_{-1}, respectively. We then define three complementary conditions:

- O if $|d_i| = 0$;
- C_i if $0 < |d_i| \leq \gamma_i$;
- $\overline{C_i}$ if $|d_i| > \gamma_i$.

We can now define the six-context table 12.1, and examples for their associated heuristic polynomial predictors.

Table 12.1: Summary of the six-context Table.

n(ame)	#	$\lvert d_{-1}\rvert$	$\lvert d_0\rvert$	$d_{-1}.d_0$	(δ, λ)
f(lat)	0	O	O	O	$(0,1)$
c(alm)	1	C_{-1}	C_0	any	$(2,5)$
m(ove)	2	C_{-1}	$\overline{C_0}$	any	$(0,1)$
r(est)	3	$\overline{C_{-1}}$	C_0	any	$(0,2)$
t(ake)	4	$\overline{C_{-1}}$	$\overline{C_0}$	> 0	$(1,3)$
j(ump)	5	$\overline{C_{-1}}$	$\overline{C_0}$	< 0	$(0,1)$

The six contexts form a partition, i.e. they are mutually exclusive, and cover all possible options for a hybrid dynamical system. They are illustrated in figure 12.1. Their names represent their behavior. For instance, the flat context addresses steady signals, for which a mere zeroth-order hold suffices, hence $P_{(0,1)}$. The calm context represents a sufficiently sampled situation, where value increments over time remain below fixed thresholds. In this case, the signal is relatively regular, and could be approximated by a quadratic polynomial, for instance $P_{(2,5)}$. For the "flat" and "jump" contexts, there is an additional procedure which consists in resetting the extrapolation to prevent inaccurate prediction. For example, when context 1 is chosen just after context 5, the quadratic extrapolation $P_{(2,5)}$ requires 5 valid samples, whereas the last 3 only are relevant.

Our two-threshold selection is relatively simple. Hence, the choice of the thresholds γ_0 and γ_{-1}, is potentially crucial. For instance, fixed values may reveal inefficient under important amplitude or scale variation of signal. Hence, we have chosen here to compute them, in a running manner, on the past frame $\{u_{1-\omega}, \ldots, u_{-3}\}$. With excessively low thresholds, high-order extrapolations would be rarely chosen, loosing the benefits of predictions. Too high thresholds would in contrast suffer from any unexpected jump or noise. As the contexts are based on

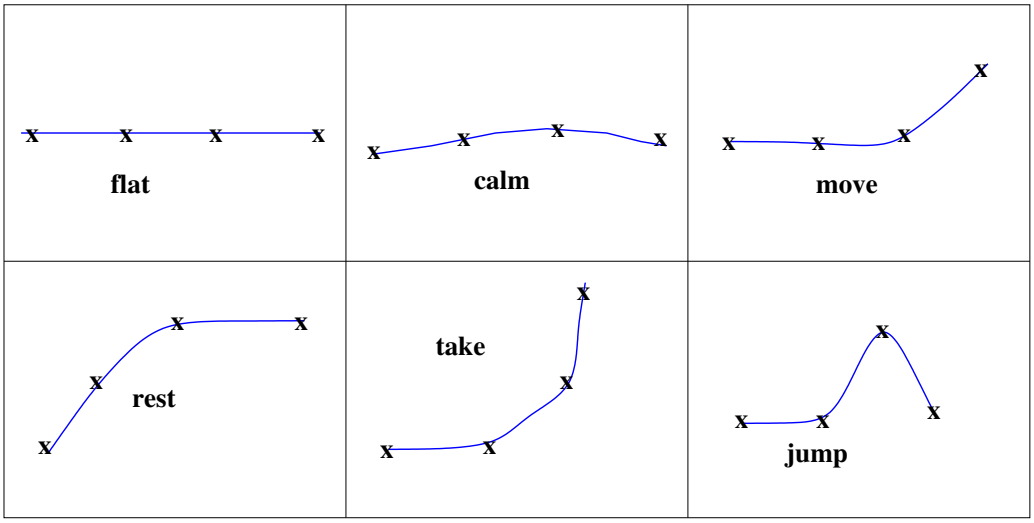

Figure 12.1: Illustration for context table in table 12.1.

backward derivatives, we have used in the simulations presented here the mid-range statistical estimator of their absolute values. This amounts to set: $\gamma_0 = \gamma_{-1} = \frac{1}{2} \max_{i \in [1-\omega,\ldots,-3]} (|u_i - u_{i+1}|)$.

12.4 Tests and results

Tests are performed on the same platform of section 11.4, with 16 GB RAM and 2 "Intel Xeon" processors, each running 8 cores at 3.1 GHz.

The simulation reference is built in the same way as in section 11.4.2, respecting the relative error Er, defined in (11.1). Besides, the split CFM-engine model, described in chapter 7 and section 8.3, is simulated in xMOD with the "RCosim" approach (defined in chapter 11).

12.4.1 Effect of the context-based extrapolation on accuracy

To explore the effect of extrapolation on accuracy, the communication step has been set to 250 µs in a first set of experiments. This value has been chosen to provide acceptable results for the accuracy ($Er \approx 1\%$), while being large enough to make extrapolation useful.

The tests show that performing only a fixed polynomial prediction on the engine model fails, with integration errors larger than for the reference simulation. This is due to the hybrid nature of the model, for which the extrapolation failures are caused by discontinuities, and also by sharp variations of some variables at specific instants. These cases totally waste the gain in precision due to successful extrapolation in the other parts of the state trajectories.

In contrast, using the context-based polynomial predictor, the outputs of the simulation are always closer to the reference trajectory than those computed when considering the inputs hold as constant (see figure 12.2).

Figure 12.3 shows that using context-based extrapolation, the prediction step is discarded when there is a discontinuous behavior in the signal, and that the degree of the predictor is adapted according to the signal slope.

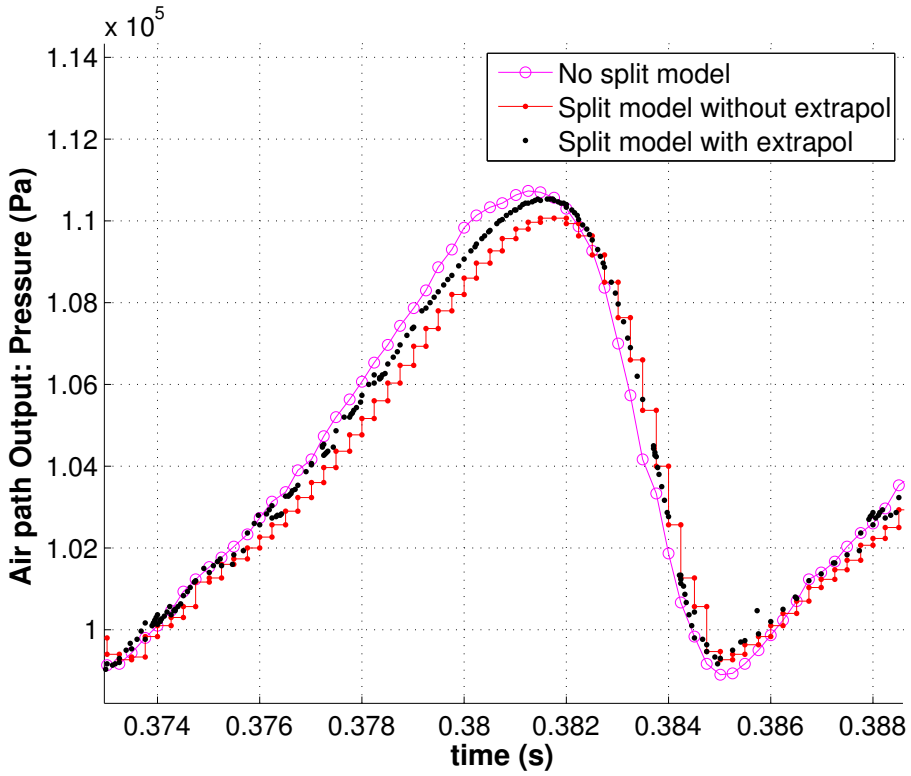

Figure 12.2: Air path output: pressure.

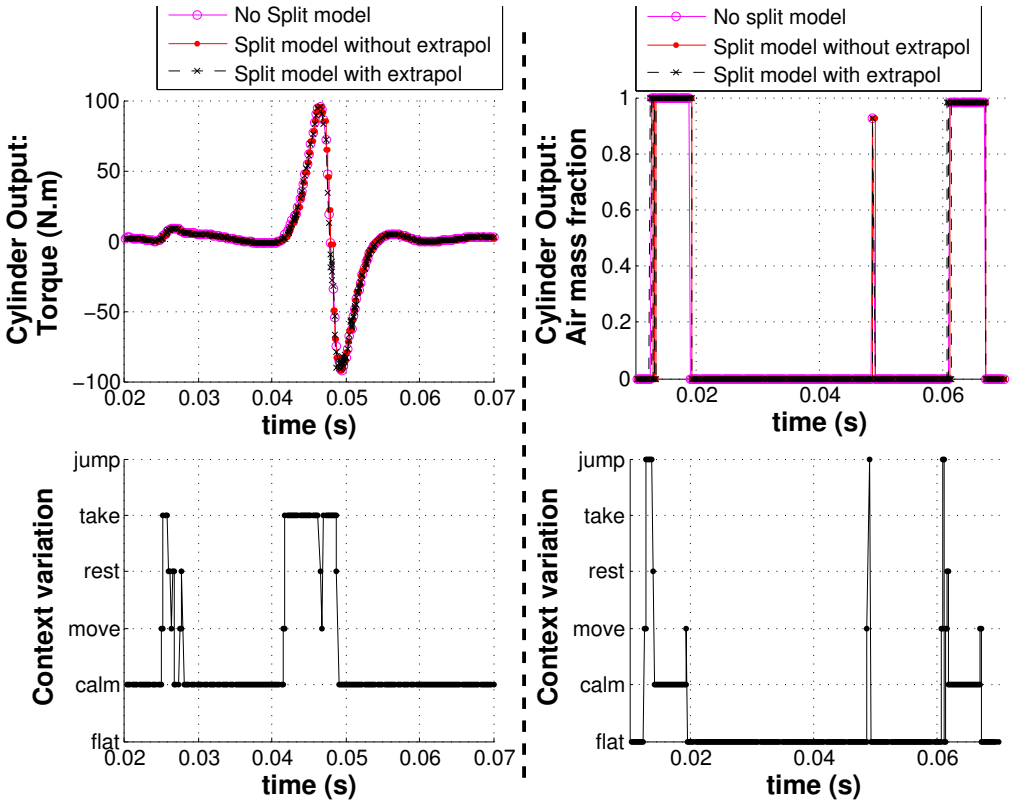

Figure 12.3: Context behavior during simulation.

The cumulative relative integration error on a long simulation run is computed in table 12.2. It shows that the context-based extrapolation efficiently decreases this error for the chosen

variables, for example by 63 % for the temperature and by 72.5 % for the fuel density.

Table 12.2: Relative integration error.

Outputs	Er(%) w/o extrapolation	Er(%) w/ extrapolation
Pressure	0.499	0.304
Temperature	0.511	0.19
Air density	0.784	0.31
Fuel density	3.55	0.978
Burned gas density	4.99	3.47

12.4.2 Effect of the context-based extrapolation on simulation time

The ultimate objective of extrapolation is to decrease the simulation time by stretching out the synchronization interval, while keeping the relative integration error Er inside predefined bounds. Indeed, widening the communication step from 100 µs to 250 µs without extrapolation (see figure 12.4) saves time but increases the error (e.g. 6.97 % for the burned gas density and 340.5 % for the fuel density).

Using the extrapolation for the 250 µs step fortunately decreases the relative error to values close to, or below, those measured for the 100 µs step with frozen inputs.

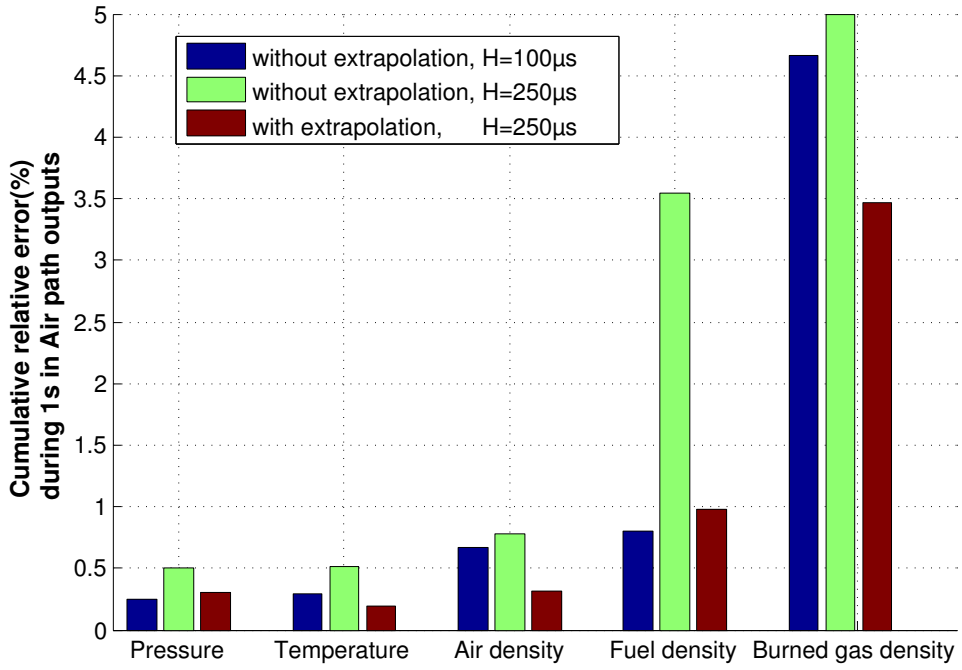

Figure 12.4: Cumulative relative error using different communication steps.

Table 12.3 shows the simulation speed-up compared with the single-threaded reference. First, note that when splitting the model into 5 threads integrated in parallel on 5 cores, the speed-up is supra-linear w.r.t. the number of cores. Indeed, the containment of events detection and handling inside small subsystems allows for solvers accelerations, enough to over-compensate the multi-threading costs. Secondly, it appears that combining the enlarged communication

step and the context-based extrapolation, the 10 % extra speed-up is reached without loss for the relative error. Even more surprising, using the extrapolation slightly speeds up the simulation, possibly because the inputs shaped by the predictor enables a faster convergence of the solver step.

Table 12.3: Simulation speed-up.

Communication time	100 µs	250 µs	
Extrapolation	No	No	Yes
Speed-up	8.9	10.01	10.07

12.5 Conclusion

This chapter proposes an approach of stretching out the communication steps while keeping a predefined integration precision. Rather than using costly small integration and communication steps, it uses extrapolations of the behavior of the models over the synchronization intervals. Test results on a hybrid dynamical engine model show that well-chosen context-based extrapolation allows for an effective speed-up of the simulation with negligible computing overheads.

This work shows that properly-chosen context-based extrapolation, combined with model splitting and parallel integration (refined scheduling co-simulation "RCosim"), can potentially improve the speed/precision trade-off needed to eventually reach real-time simulation. However, the accuracy could be widely improved by accessing on the current input derivatives of the models, since the future behavior of the signals can be then known. This is the case for the FMI for Co-Simulation, and it would be highly useful to also integrate this feature in the FMI for Model Exchange.

We remind that, the proposed polynomial extrapolation is based on fixed synchronization intervals equal to H. Future enhancements can consider communication step-size control, for which the error analysis and estimation can be inspired by [118].

Part IV

Conclusion

Chapter 13

Conclusion

13.1 Summary

The complexity of mechatronic systems is due to different factors. The vehicle development, for instance, depends on a high number of subsystems such as powertrain, alternative fuels, control systems, driver assistance systems, regulation, vehicle safety, etc. The variety and complexity of the interactions between these components require constant exchanges between the expert teams.

The system simulation approach allows for the complexity of the overall dynamic system to be mapped with its environment through virtual development tools, where the components which are available as real hardware are directly connected to the system models. Besides, co-simulation allows early predictions and design decisions of complex systems as well as the integration of real-time systems into the system simulation.

However, several issues concerning the co-simulation have to be solved to guarantee the precision of the simulation results, such as accurate data exchange, coupling different dynamic systems' behavior, real-time constraints and models computational complexity.

Nowadays, phenomenological models cannot be simulated in HIL because they involve high computations. To meet real-time constraints, engineers spend time to reduce the model representativeness from the 0D phenomenological form to a simplified quasi-static form. The objective of this thesis is to improve this validation stage by keeping such representativeness in HIL simulation. For this aim, we propose some methods that speed-up the simulation without loosing in results accuracy.

The proposed approaches developed in this thesis are structured around a 0D phenomenological internal combustion engine case study developed in IFP Energies nouvelles (chapter 7). First, a model decomposition from a physical point of view (chapter 8) is presented and applied in the context of the modular co-simulation. The approach of the thread level parallelism shows the interest of splitting models when dealing with complex hybrid systems. In fact, test results shows that the major cost in numerical integration lies in the computation of the derivatives and in the events detection and location (root-finding). Hence, they can be reduced thanks to decoupling sub-models.

After that, in chapter 9, a convergence analysis of the different models of computations used for the modular co-simulation in the context of IFP Energies nouvelles (more precisely in the xMOD tool) is performed to determine the major actors on the simulation errors. We show analytically that the error is related directly to the numerical solver (integration step, order),

the models coupling and the communication step. The effect of the numerical solvers is already testified in chapter 8.

For the coupling between models, we first proposed, in chapter 10, a model decomposition based on the structural analysis of the system, i.e. on different incidence matrices of the states and the events. This method is interesting especially for systems with no obvious or intuitive partition. Then, in chapter 11, we propose an approach to schedule in a refined way the different threads (models) onto the different cores. This new model of computation schedules only the inputs/outputs operations, then run in parallel the operation of states computation (the numerical integration). This method shows through the split case study, that it enhances a supra-linear speed-up, already enabled by the modular co-simulation, and at the same it improves the accuracy of the simulation. Besides, this approach allows for the reduction of the time spent by engineers to determine how to split a system.

Finally, for the communication step, we propose in chapter 12, a polynomial prediction of the models' inputs based on predefined contexts. Test results show that using contexts is an important added value for the extrapolation when dealing with hybrid systems. Besides, thanks to this new technique, the communication step between the loosely coupled models can be slackened and stretched to speed up the simulation and, at the same time, the accuracy of the results can be improved with a negligible cost in the extrapolation.

13.2 Perspectives

The following research directions represent possible extensions of the present work:

Communication step-size control

In practical applications, current co-simulation set-ups use a constant communication grid H, e.g. fixed by the FMI for Model Exchange 1.0 specification, since there is no possible rollbacks (the states cannot be saved). Further improvements are expected from adaptive communication step-sizes, allowed with the recent version of the specification FMI for Model Exchange 2.0. It is expected to better handle the various changing dynamics of the models [119].

In fact, the size of the communication steps has a direct impact on the simulation errors (summarized in section 9.5), and effective communication step control should rely on on-line estimations of the errors induced by slackened exchange rates (a first proposal was detailed in section 9.6). Indeed the stability of multi-rate simulators with adaptive steps needs to be carefully assessed, for example based on recent work on errors propagation inside modular co-simulations [86].

Multi-rate refined scheduling co-simulation

The refined scheduling co-simulation "RCosim" (detailed in chapter 11), treats the case where the co-simulation uses a common communication step-size H that is shared by all the models. In fact, all the models read their inputs and update their outputs at the same communication points, that are multiple of H. Future enhancements aim to generalize this proposed technique of refined scheduling to the multi-rate case. Indeed, this will allow to avail the benefit of both multi-rate co-simulation and "RCosim" approach.

Context-based extrapolation

Future work intend to improve the presented context-based extrapolation algorithm (detailed in chapter 12), to make it more subtly aware of data freshness and even more decrease the prediction induced integration errors. Another possibility is to process the input signals to separate them into simpler components, easier to predict with different predictors, and to cope with noise. When it comes to polynomials, wavelet pre-processors [120] could be useful, as they play an important role in polynomial model fitting.

Quantized state solvers

This thesis focused on numerical solvers based on time discretization (studied in section 4.2. Besides, investigation was also made around solvers based on state quantization (see section 4.3) to compare their efficiency to time discretization, since QSS are known suitable for discontinuous ODEs [121]. Nevertheless, current results on QSS are mostly applied to academic examples and we encountered difficulties to test it on industrial examples such as the engine model case study. In fact, the main obstacle was the mandatory translation of the model from the Modelica language to the μ-Modelica language. Firstly, it was made by hand since the automatic translator is currently in progress and so not already available for the end user. For instance, for the mono-cylinder engine model, the number of code lines of the μ-Modelica translated form is about one thousandth. Secondly, the expressivity of the μ-Modelica language is restricted compared to the Modelica language, so that the engine model were not able to be modeled correctly using the μ-Modelica language. Anyway, as QSS solvers seem to have a promising potential for the integration of dynamical systems with many discontinuities, progress in QSS theory and associated tools deserve to be closely followed.

Epilogue

As a final word, the thesis work provides effective and already usable solutions for the initial challenging topic. It contributes for both scientific and technological progress, and the objectives are met by supplying methodological advancements for the parallel co-simulation of complex systems, as well as a practical solutions which can be used from now by engineers.

Although our case study presents an obvious and effective natural partitioning, the proposed methodology can be easily applied to other complex hybrid dynamical systems. Indeed using variable step solvers, even for the real-time framework, is the first key step beyond the HIL state of the art. Understanding what are the main bottlenecks for achieving solvers speed-ups is the second key step, providing the directions to find effective partitioning rules and tools. Finally refined scheduling and extrapolation allows for enhanced numerical integration speed-ups, so that reaching real-time high fidelity simulation, e.g., for the automotive framework, becomes feasible.

The methodologies and software tools developed in the thesis are expected to be quickly exploited in industrial developments, in particular for the design of new gasoline engines with extra low emission levels.

Appendices

Annexe A

Résumé détaillé en Français

A.1 Introduction générale

Avant-propos

Ce document synthétise les travaux accomplis durant les trois ans de thèse de doctorat intitulée "Simulation temps-réel distribuée de modèles numériques : application au groupe motopropulseur", sous la direction de Daniel SIMON[1] *et Mongi BEN GAID*[2]*. Cette thèse a été financée par* IFP Energies nouvelles *et réalisée dans la direction Technologie, Informatique et Mathématiques Appliquées à la suite de la thèse de Cyril FAURE [1]. Les travaux de thèse ont donné lieu à plusieurs publications résumées à la fin de ce manuscrit.*

A.1.1 Contexte général

Un défi majeur du $21^{ème}$ siècle est de réussir la transition énergétique, d'une économie qui est actuellement basée sur l'énergie fossile, à une économie qui s'appuie sur les énergies renouvelables et l'efficacité énergétique. Ce défi concerne l'ensemble du cycle énergétique : la production, le transport ainsi que la consommation.

Le secteur des transports consomme des quantités importantes d'énergie. Il est principalement tributaire du pétrole, une ressource limitée dont le prix ne cesse d'augmente en raison d'une disponibilité en diminution, et qui est prévu de disparaître d'ici la fin du siècle. Réduire la consommation de carburant et la diversification des sources d'énergie sont des défis majeurs dans ce domaine.

D'autre part, le réchauffement de la planète et les changements climatiques font actuellement partie des principales préoccupations des gouvernements du monde entier, conduisant ainsi à prendre des mesures importantes et restrictives afin de limiter les émissions de polluants.

Dans ces perspectives, le secteur automobile voit sa réglementation devenir de plus en plus stricte vis à vis de la réduction de la consommation de carburant et des émissions de polluants. Les nouveaux véhicules doivent se conformer à ces normes pour qu'ils puissent être mis sur le marché. Par exemple, la norme européenne sur les émissions vise à réduire les émissions nocives, notamment les oxydes d'azote (NOx) et les particules fines (PM) comme cela est illustré dans

[1] INRIA : `http://www.inria.fr/`

[2] IFP Energies nouvelles : `http://www.ifpenergiesnouvelles.fr/`

la figure A.1. Ces exigences renforcent la nécessité d'adapter et de concevoir rapidement de nouveaux modèles de moteurs ainsi que des stratégies de contrôle connexes. Ceci implique l'utilisation de plusieurs technologies augmentant ainsi le nombre d'actionneurs à contrôler.

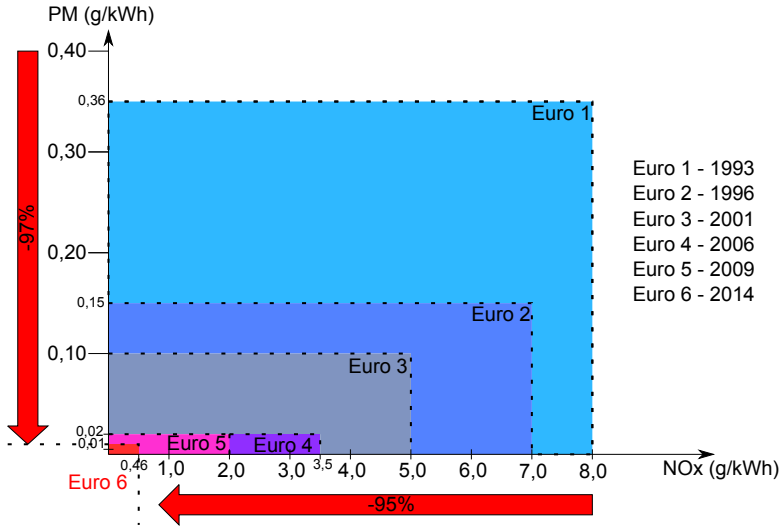

Figure A.1 – Norme européenne sur les émissions polluantes Euro 6.

Les automobiles sont des exemples typiques de systèmes cyber-physiques, où l'énergie chimique (essence, diesel, éthanol, etc.) ou électrique est convertie en une énergie cinétique. Les contrôleurs électroniques et les réseaux, présents dans le véhicule, interagissent avec les différents composants du véhicule, qui sont des sous-systèmes de nature multi-physique (chimique, mécanique, thermodynamique, électrique, etc.), et dont la conception implique des équipes pluridisciplinaires.

Lors de la phase de conception, il s'est avéré que la simulation est une étape incontournable pour la validation des prototypes. En effet, les simulations numériques permettent l'évaluation préliminaire, le réglage et éventuellement la re-conception, des solutions proposées avant leur mise en oeuvre, réduisant ainsi les risques. Pour s'assurer de l'exactitude des résultats, ces simulations nécessitent des modèles de grande fidélité pour décrire les différents composants ainsi que leurs interactions.

A.1.2 Description du problème

Actuellement, la modélisation des systèmes cyber-physiques en général et les véhicules automobiles en particulier, au moyen de modèles à haut niveau de représentativité est un défi très ambitieux et difficile à relever. Un des problèmes rencontrés réside dans la diversité des environnements de modélisation et de simulation, utilisés par les différentes équipes impliquées. Ceci est dû au fait que chacune préfère utiliser son environnement habituel bien adapté à sa spécialité (langage de modélisation, bibliothèques, solveurs, coûts, etc.). La spécification FMI a été proposée pour résoudre ce problème [2].

Un deuxième problème est lié directement au coût exorbitant du temps de calcul observé durant l'exécution des modèles à haut niveau de représentativité. La principale raison est que actuellement, la majorité des logiciels de simulation système ne sont pas en mesure d'exploiter les processeurs multi-coeurs, puisqu'ils utilisent le plus souvent des solveurs basés sur des équations différentielles (algébriques et ordinaires) séquentielles.

Cependant, l'amélioration de la puissance de calcul des processeurs actuels provient plus de l'augmentation du nombre de coeurs par processeur que de l'augmentation de la fréquence des coeurs. Pour résoudre ce problème, les approches de co-simulation peuvent apporter des améliorations significatives en permettant ainsi de simuler des modèles provenant de différents domaines et de valider aussi bien les comportements individuels que le comportement global [3]. Les simulateurs peuvent être exportés à partir de leurs outils de développement d'origine sous la forme de FMUs, puis importés dans un environnement de co-simulation, pour qu'ils puissent coopérer à l'exécution grâce aux fonctionnalités du FMI.

La modélisation ainsi que l'intégration numérique induisent des approximations, par conséquent il est d'abord nécessaire de trouver un moyen de satisfaire le compromis entre la vitesse de simulation et la précision des résultats. En définitive, la simulation des modèles physiques tiendra compte des contraintes temps-réel mise en place par l'interaction avec les composants réels. Ces interactions entre les composants réels et les composants simulés définissent la simulation HIL (Hardware-In-the-Loop). Les modèles (composants simulés) sont destinés à valider les contrôleurs (composants réels), c'est pour cela que l'interaction entre le monde simulé et le monde réel doit être cohérente, c'est-à-dire que le temps simulé et le temps-réel doivent concorder à certains points précis [4].

Cependant, l'utilisation de modèles de haute précision en simulation HIL, pour la validation des unités de contrôle, est souvent entravée par les limitates de performance des méthodes habituelles de simulation, c'est-à-dire mono-coeur et mono-solveur. En effet, la simulation des systèmes complexes est très coûteuse en matière de temps de calcul et l'utilisation d'un seul processeur ne permet pas de simuler en temps-réel. C'est pour cela que le calcul parallèle pourrait être la solution pour s'assurer du respect des contraintes temps-réel, au moyen de méthodes de décomposition de modèles et de méthodes de simulation parallèle des différents sous-modèles créés.

Les dépendances de données, dues au couplage des sous-modèles, produisent des périodes d'attente entre tâches, donc des temps d'inactivité pour le processeur, ce qui réduit l'efficacité du parallélisme apporté par la plate-forme multi-coeurs. Par conséquent, ces contraintes de dépendance doivent être relâchées autant que possible pour améliorer le rendement du parallélisme, tout en évitant de produire de trop grandes erreurs numériques dans les résultats de simulation. En effet, une synchronisation minimale doit être garantie entre les sous-modèles pour limiter ces erreurs. Par conséquent, le respect de ce compromis peut dépendre d'une décomposition efficace du modèle qui permettrait de découpler dans la mesure du possible les sous-modèles. Ainsi, le relâchement des dépendances de données pourrait permettre d'élargir l'intervalle de synchronisation pour accélérer au maximum le temps de simulation tout en préservant la qualité des résultats.

A.2 Contributions de la thèse

Cette thèse étudie et propose des méthodes analytiques et expérimentales qui visent la co-simulation temps-réel distribuée de modèles dynamiques hybrides avec des synchronisations relâchées. Le terme "distribué" fait référence dans cette thèse à la répartition des tâches (ou modèles) sur une architecture parallèle (multi-coeurs). En effet, cette thèse a pour objectif de définir des solutions pour exploiter plus efficacement le parallélisme fourni par les architectures multi-coeurs en utilisant de nouvelles méthodes d'allocation des ressources. Ces solutions visent à valider des modèles phénoménologiques complexes directement par la simulation HIL.

A.2.1 Cas d'étude : Moteur à combustion interne

Description du moteur

Dans cette étude, un moteur à allumage commandé "RENAULT F4RT" a été modélisé avec 3 gaz (air, carburant et gaz brûlés). Il s'agit d'un quatre cylindres avec une pompe d'injection en ligne et une cylindrée de 2 L. La combustion est considérée comme homogène. Le boucle d'air utilise un turbocompresseur avec une turbine à simple spirale contrôlée par une "wastegate", un papillon d'admission et un échangeur de chaleur en aval du compresseur. Pour finir, ce moteur est équipé de deux distributions variables, pour les soupapes d'admission et d'échappement, afin d'améliorer l'efficacité du moteur (performance, carburant et émissions). La puissance maximale est d'environ 136 kW à 5000 tr/min.

Description des modèles de combustion

Deux types de modèles de combustion ont été utilisés dans cette étude, "Wiebe" et "CFM". Ces modèles partagent des équations thermodynamiques basiques, comme l'équation de conservation de masse, l'équation de gaz parfait, l'équation de conservation d'énergie, etc. La principale différence réside dans les termes d'échange de chaleur lors de la combustion c'est-à-dire dans la manière dont est gérée la combustion.

- Le "CFM" est un modèle 1D phénoménologique, développé à IFP Energies nouvelles [98] à partir de la réduction du modèle 3D "ECFM" [99]. Le taux de consommation de carburant dépend de la surface de la flamme laminaire, calculée grâce à la vitesse de la flamme et à l'énergie cinétique turbulente. Un seul paramètre lié à l'énergie cinétique turbulente est accordé pour l'étalonnage de combustion, les autres restent constants. Sa modélisation combine une bonne représentativité des phénomènes physiques avec des performances CPU raisonnables. Grâce à ces caractéristiques, ce modèle peut être intégré dans un simulateur de moteur complet pour la conception de l'architecture du moteur ainsi que de ses stratégies de contrôle [100]. En matière de complexité, le modèle moteur "CFM" a 118 variables d'état continues $\boldsymbol{X}$, 398 indicateurs d'événements (discontinuités) $\boldsymbol{Z}$, 1466 équations et 7907 variables (dont 1979 inconnues).
- "Wiebe" est un modèle semi- physique, basé sur une approche analytique du dégagement de chaleur lors de la combustion [97], qui présente moins de complexité . Il est basé sur une combinaison d'approches physiques et d'identifications. Ses paramètres sont calés et optimisés à l'aide des résultats expérimentaux réalisés avec un modèle plus complexe. Le principal avantage de ce modèle est qu'il prend en considération le comportement du moteur avec une échelle de temps basée sur l'angle de vilebrequin, ce qui n'est pas le cas des modèles basés sur des tables de correspondance. En matière de complexité, le modèle moteur "Wiebe" a 78 variables d'état continues $\boldsymbol{X}$, 420 indicateurs d'événements (discontinuités) $\boldsymbol{Z}$, 1334 équations et 7767 variables (dont 1922 inconnues).

Modélisation et simulation du moteur

Le modélisation du moteur F4RT a été réalisée grâce à librairie "ModEngine" [101]. "ModEngine" est une librairie Modelica [102] qui permet la modélisation des moteurs diesel et des moteurs à essence. Par la suite, le modèle est importé dans l'outil xMOD en utilisant la fonctionnalité export du FMI [103] disponible dans Dymola. Plus précisément, la spécification FMI

décrit l'interface logicielle d'un système hybride décrit par des équations différentielles ordinaires (ODEs). Enfin, le modèle moteur peut être connecté à son contrôleur développé dans Simulink grâce aux capacités d'intégration de xMOD.

A.2.2 Décomposition de modèle d'un point de vue physique

Introduction

Dans l'approche systémique, le système complexe est vu comme un ensemble de sous-systèmes. Étant donné que notre approche s'intéresse au parallélisme au niveau du "thread" (fil d'exécution), chaque sous-système est alors associé à un "thread". Les connexions entre les sous-systèmes représentent les différents flux de données échangés entre eux, alors que d'un point de vue tâches informatiques, ces dépendances définissent l'ordre d'exécution entre les "threads".

Afin de réaliser une simulation multi-coeurs, nous proposons une méthode de décomposition basée sur la connaissance du comportement physique du système. Cette approche est appliquée sur le modèle moteur, décrit précédemment, et peut être reproduite sur d'autres systèmes dynamiques hybrides complexes.

Description de la méthode

La partitionnement naturel et intuitif du modèle moteur est réalisé en séparant les quatre cylindres de la boucle d'air (AP), puis en isolant les cylindres (C_i, pour $i \in [1,2,3,4]$) l'un de l'autre. D'un point de vue thermodynamique, les cylindres sont couplés de façon lâche, mais un échange mutuel des données existe encore entre eux et la boucle d'air. La dynamique de la boucle d'air est lente (elle produit des sorties, destinées aux cylindres, qui varient lentement, par exemple la température) comparée à celle des cylindres (ils produisent des sorties, destinées à la boucle d'air, qui varient rapidement, par exemple le couple). De plus, contrairement aux sorties des cylindres, la plupart des sorties de la boucle d'air ne sont pas fonction directes des entrées de la boucle d'air (elles sont appelées sorties NDF, définies dans la section 9.2.1). Par conséquent, le choix de l'ordre d'exécution des sous-modèles est fait de la boucle d'air vers les cylindres (conformément à l'analyse décrite dans la section 9.3.2).

Le modèle est divisé en 5 composants et commandé par un contrôleur basique noté CTRL, comme c'est illustré dans la figure A.2. Il rassemble 91 entrées et 98 sorties, indépendamment du modèle de combustion choisi (Wiebe ou CFM).

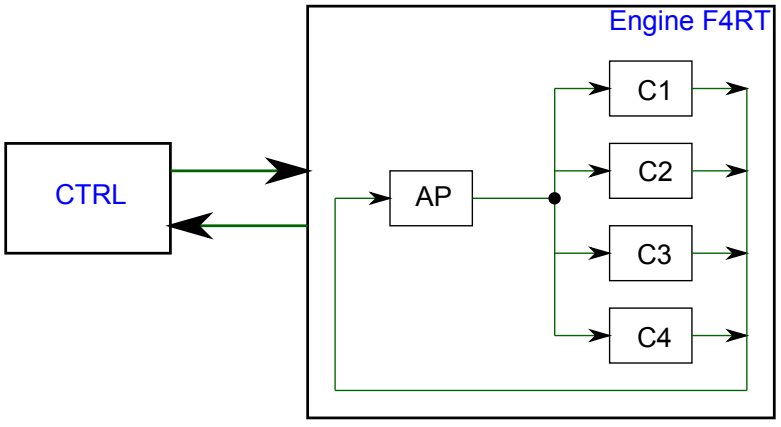

Figure A.2 – Modèle moteur partitionné en 5 composants.

Cette étude compare les performances de simulation, en observant le compromis entre la vitesse et la précision de la simulation, pour les approches suivantes :

- La simulation du modèle moteur en un seul "thread" et en utilisant un seul solveur. Ce cas correspond à la référence pour les évaluations de précision et d'accélération ;
- La co-simulation modulaire du modèle partitionné avec respect des dépendances de données. Il s'agit de la version standard de la co-simulation modulaire, notée "sv-MCosim", où l'ordre d'exécution est fixé des modèles lents vers rapides. Pour le cas d'étude, tous les cylindres doivent attendre l'exécution de la boucle d'air ;
- La co-simulation modulaire du modèle partitionné sans respect des dépendances de données. Il s'agit de la version étendue de la co-simulation modulaire, notée "ev-MCosim", où toutes les dépendances de données sont relâchées (en utilisant les dernières données disponibles). Pour le cas d'étude, la boucle d'air et tous les cylindres sont intégrés en parallèle à chaque pas de communication sans période d'attente.

Ces méthodes sont illustrées dans la figure A.3, où DT est le temps d'exécution pendant un pas de communication. DT regroupe l'intégration des modèles dans les blocs X_i (par exemple X_{AP} pour la boucle d'air AP) et les mises à jour des Entrées/Sorties dans les blocs IN/OUT.

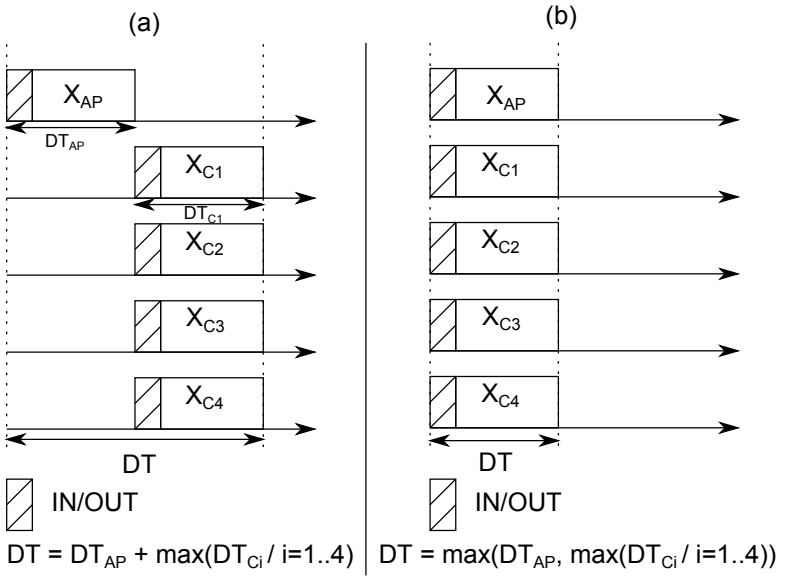

Figure A.3 – (a) méthode sv-MCosim ; (b) méthode ev-MCosim.

Validation expérimentale

Dans les essais suivants, les simulations du moteur F4RT, avec les deux modèles de combustion "Wiebe" et "CFM", sont réalisées sous xMOD. Comme première approche, l'idée est de comparer le solveur à pas de temps variable LSODAR au solveur à pas de temps fixe RK4. Le pas d'intégration du RK4 est choisi très petit (50 µs), qui est considéré comme référence par les ingénieurs de modélisation et de simulation. La validation est basée sur des quantités d'intérêt telles que les pressions d'admission et d'échappement, l'AFR et le couple.

Avant d'utiliser le solveur LSODAR localement dans chaque sous-modèle (thread), un important travail préliminaire est réalisé pour rendre LSODAR "thread safe", afin de l'intégrer dans le cadre du "FMI for Model Exchange" de xMOD.

La figure A.4 illustre la pression de collecteur d'admission ainsi que le couple pendant 1 cycle moteur, qui correspond à 2 tours de vilebrequin (en utilisant le modèle Wiebe avec une vitesse égale à 2500 tr/min). Ces sorties sont calculées en utilisant à la fois LSODAR avec un pas de communication égale à 500 µs et une tolérance égale à 10^{-5}, et RK4 avec un pas d'intégration égale à 50 µs, afin d'assurer une précision acceptable. En effet, l'erreur entre la sortie de la pression est inférieure à 0.3 % et celle du couple est inférieur à 0.5 %.

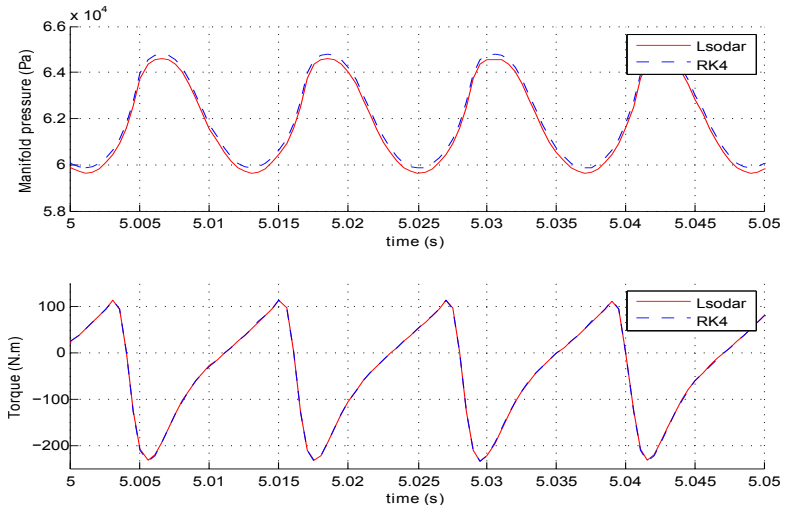

Figure A.4 – Quelques sorties intégrées avec les solveurs RK4 et LSODAR.

Avec LSODAR, la bornitude des erreurs d'intégration est assurée. Cependant, le temps d'exécution est 4 fois plus long, avec une tolérance de 10^{-4} et 6 fois plus long avec une tolérance de 10^{-5}.

Après une analyse approfondie de l'exécution du solveur, la lenteur peut s'expliquer par la présence d'un grand nombre de discontinuités qui diminue l'avantage de la vitesse des solveurs à pas de temps variable. En fait, les discontinuités impliquent un calcul coûteux de la fonction *zero-crossing* dans (3.6), utilisée pour la *détection* et la *localisation* des événements, ainsi que le redémarrage du solveur pour la *gestion* des événements.

Puisqu'en général les événements sont liés uniquement à l'évolution d'un sous-ensemble du vecteur d'état, la décomposition du modèle moteur permet à chaque sous-modèle d'être intégré par son propre solveur, en évitant ainsi les interruptions provenant d'événements d'autres sous-modèles. En effet, la phase de combustion détient la plupart des événements, qui sont localisés dans la chambre de combustion du cylindre. Le solveur peut donc les traiter localement pendant cette phase, puis agrandir son pas d'intégration jusqu'au prochain cycle.

La première étape consiste à comparer le temps d'exécution en mono-coeur entre le modèle original mono-thread et le modèle partitionné multi-thread, afin de voir uniquement l'impact du relâchement des événements sur l'accélération du solveur LSODAR, sans l'effet de la parallélisation multi-coeurs.

- Résultat 1 : Nombre de discontinuités

La décomposition du modèle permet de diminuer le nombre de discontinuités traitées par le solveur. En effet, les tests réalisés pendant 0.3 s montrent que le modèle non décomposé présente 851 événements, tandis que le modèle décomposé présente en moyenne 203 événements par cylindre et 119 pour la boucle d'air. La figure A.5 illustre ce résultat durant 2 cycles moteur.

- Résultat 2 : Taille du pas d'intégration

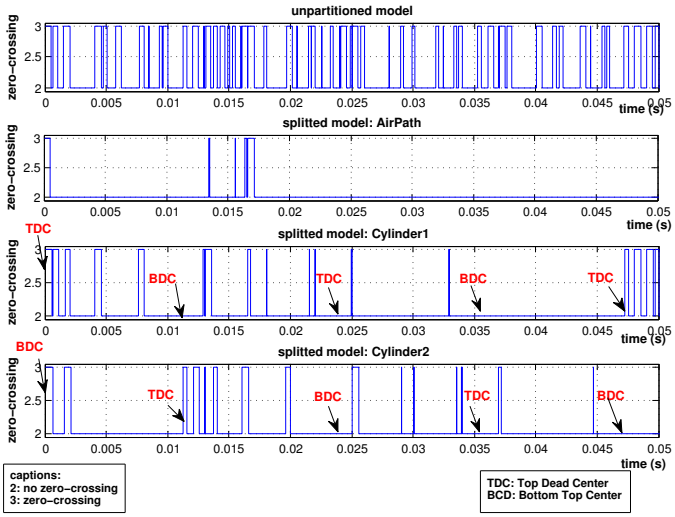

Figure A.5 – Nombre de discontinuités par modèle.

Grâce à la réduction du nombre d'événements par sous-modèle, le nombre d'interruptions pendant l'intégration est réduit aussi. Cela permet donc au solveur d'augmenter son pas de temps comme c'est illustré dans la figure A.6. En effet, dans le modèle global d'origine, la valeur maximale et la valeur moyenne de la taille du pas d'intégration sont d'environ $h_{\max} = 422\,\mu s$ et $h_{\text{moy}} = 148\,\mu s$ alors que pour le modèle partitionné, la taille du pas peut atteindre une valeur maximale de $h_{\max} = 500\,\mu s$ et une valeur moyenne d'environ $h_{\text{moy}} = 215\,\mu s$ pour les cylindres et $h_{\text{moy}} = 229\,\mu s$ pour la boucle d'air.

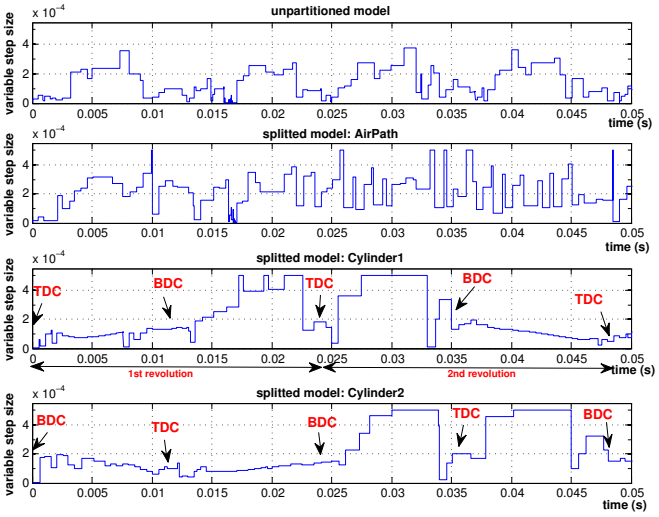

Figure A.6 – Comportement du pas d'intégration par modèle.

- Résultat 3 : Temps d'exécution

Les résultats 1 et 2 entraînent une accélération du temps d'exécution, d'environ 1.98 sans l'apport de la parallélisation multi-coeurs.

L'intérêt maintenant porte sur la simulation parallèle du modèle en utilisant un PC multi-coeurs. Les tests sont réalisés en utilisant le solveur RK4 sur le modèle moteur CFM. Ce modèle est exécuté comme le montre la figure A.7 en utilisant en premier lieu 2 puis 4 coeurs. Ensuite, les dépendances de données entre les sous-modèles sont relâchées pour que le modèle puisse être exécuté sur 5 coeurs.

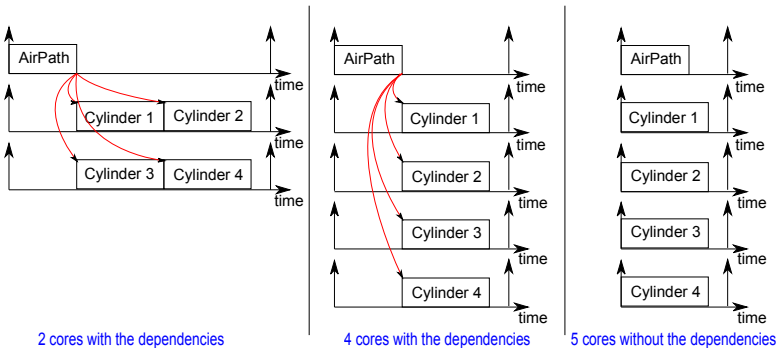

Figure A.7 – Distribution du modèle moteur.

Les résultats des tests montrent que l'accélération du temps d'exécution, par rapport au cas mono-coeur, est d'environ 1.77 pour une simulation parallèle avec 2 coeurs et de l'ordre de 3.15 avec 4 coeurs. Finalement, l'accélération atteint environ 3.9 avec l'utilisation de 5 coeurs. La question qui se pose dans ce cas est alors "quel est le meilleur compromis concernant le temps d'exécution et la précision : le relâchement des dépendances compensé par une intégration avec un pas égale à 50 µs, ou bien le maintien de ces dépendances avec un choix de pas d'intégration plus large, égale à 100 µs".

Le tableau A.1 montre que le choix de casser les dépendances de données entre la boucle d'air et les quatre cylindres, en utilisant RK4 avec un pas de temps $h = 50\,\mu s$, permet d'avoir moins d'erreurs.

Table A.1 – Erreurs d'intégrations pour quelques sorties.

	Cas 1[1] :Er(%)	Cas 2[2] :Er(%)
AFR	1.02	0.53
Pression d'admission	0.47	0.42
Vitesse du compresseur	0.65	0.50
Pression d'échappement	0.93	0.94
Couple	7.32	0.55

En ce qui concerne le temps d'exécution, ce même choix (cas 2) est plus rapide, l'accélération est d'environ 2.06. Nous pouvons conclure alors que l'exécution du moteur CFM en multi-coeurs est préférable vis à vis du temps d'exécution et de la précision, lorsque les dépendances entre la boucle d'air et les cylindres sont relâchées, avec un pas de temps égal à 50 µs, que lorsque les dépendances sont respectées avec un pas de temps plus large égal à 100 µs.

Conclusion

Ces résultats montrent l'importance et l'impact du réglage de certains paramètres (pas d'intégration, pas de communication, modèle de calculs, etc.) sur la précision des résultats de simulation. Cela rend indispensable l'évaluation de l'erreur de simulation et l'analyse de convergence des résultats. Les résultats précédents fournissent des indices pour penser tout d'abord au pas de communication.

[1] sv-MCOSIM et h=100 µs
[2] ev-COSIM et h=50 µs

A.2.3 Étude théorique l'évaluation de l'erreur

Formalisation de la co-simulation modulaire

Pour exécuter le système en parallèle, le système dynamique hybride initial Σ' décrit dans (3.5) est partitionné en plusieurs sous-modèles. Pour simplifier, supposons que le système est décomposé en deux blocs distincts notés modèle 1 et modèle 2, dans la figure A.8. Notre approche se généralise à toute décomposition en B blocs ($b = 1, \ldots, B$).

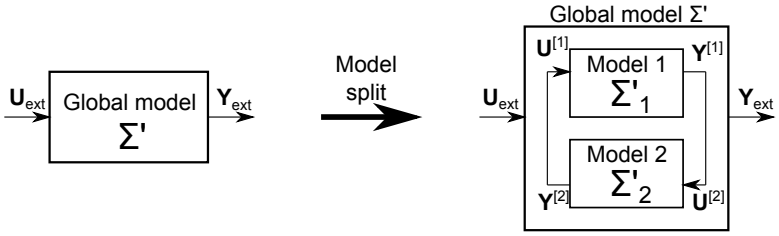

Figure A.8 – Décomposition d'un système.

Les sous-systèmes s'écrivent alors sous la forme :

$$\left\{ \begin{array}{rcl} \dot{\boldsymbol{X}}^{[1]} & = & \boldsymbol{f}^{[1]}(t, \boldsymbol{X}^{[1]}, \boldsymbol{D}^{[1]}, \boldsymbol{U}^{[1]}, \boldsymbol{U}_{ext}) \\ \boldsymbol{Y}^{[1]} & = & \boldsymbol{g}^{[1]}(t, \boldsymbol{X}^{[1]}, \boldsymbol{D}^{[1]}, \boldsymbol{U}^{[1]}, \boldsymbol{U}_{ext}) \end{array} \right. \quad et \quad \left\{ \begin{array}{rcl} \dot{\boldsymbol{X}}^{[2]} & = & \boldsymbol{f}^{[2]}(t, \boldsymbol{X}^{[2]}, \boldsymbol{D}^{[2]}, \boldsymbol{U}^{[2]}, \boldsymbol{U}_{ext}) \\ \boldsymbol{Y}^{[2]} & = & \boldsymbol{g}^{[2]}(t, \boldsymbol{X}^{[2]}, \boldsymbol{D}^{[2]}, \boldsymbol{U}^{[2]}, \boldsymbol{U}_{ext}) \end{array} \right. \tag{A.1}$$

avec $\boldsymbol{X} = [\boldsymbol{X}^{[1]}\ \boldsymbol{X}^{[2]}]^T$ et $\boldsymbol{D} = [\boldsymbol{D}^{[1]}\ \boldsymbol{D}^{[2]}]^T$.

Ici $\boldsymbol{U}^{[1]}$ sont les entrées requises par Σ'_1, directement fournies par les sorties $\boldsymbol{Y}^{[2]}$ produites par Σ'_2, (même principe pour $\boldsymbol{U}^{[2]}$ et $\boldsymbol{Y}^{[1]}$). Pour intégrer numériquement l'ensemble du système multi-variables, chacun de ces simulateurs a besoin d'échanger, à chaque pas de communication, t_{s_b}, les données requises par les autres (voir la figure A.9, $b = 1, 2$).

Pour accélérer l'intégration, les blocs parallèles doivent être aussi indépendants que possible, de sorte qu'ils sont synchronisés à un pas $H^{[b]} = t_{s_b+1} - t_{s_b}$ beaucoup plus lent que le pas d'intégration interne $h^{[b]}_{n_b}$ ($H^{[b]} \gg h^{[b]}_{n_b}$). Par conséquent, entre les points de communication, chaque simulateur intègre à son propre rythme et considère ses entrées constantes.

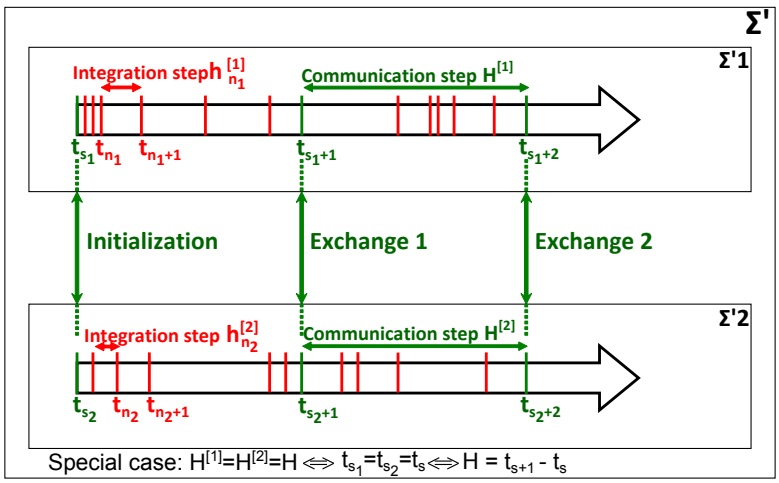

Figure A.9 – Σ' décomposé en Σ'_1 et Σ'_2 pour la simulation parallèle.

Prendre de grands pas de communication peut permettre d'accélérer l'intégration numérique, mais au prix d'une augmentation des erreurs d'intégration. Par exemple, [104] a étudié le compromis entre la stabilité et les performances de calcul dans le cadre de la co-simulation modulaire de systèmes fortement couplés. C'est pour cela que la modélisation des erreurs induites par des synchronisations relâchées présente une première étape afin de trouver les bonnes méthodes qui améliorent le compromis entre la vitesse d'intégration et la précision. Il est supposé, en premier lieu, que tous les sous-modèles partagent le même pas de communication $H = t_{s+1} - t_s$ ($H^{[b]} = H$, pour $b = 1, \ldots, B$), de sorte qu'ils lisent tous leurs entrées et mettent à jour leurs sorties à des points de communication multiples de H. Ensuite, tous les résultats sont généralisés pour le cas multi-rythmes. Par souci de simplicité, l'évaluation d'erreur théorique considère que la solution du système est assez stable et régulière par rapport aux effets des discontinuités. Ces effets sont pris en compte lors de la simulation. De plus, les erreurs d'arrondi numériques, induites par la limitation de la précision des calculateurs en virgule flottante, ne sont pas prises en compte dans cette analyse.

Analyse de convergence et évaluation des erreurs

L'analyse de convergence et l'évaluation des erreurs de la co-simulation modulaire mono-rythme, détaillées dans le chapitre 9, montrent que les états et les sorties sont bornés par l'ordre de précision du solveur, qui dépend de son ordre p et de son pas d'intégration maximum h, ainsi que du pas de communication H (voir (A.2)). Ce pas de communication peut dominer l'erreur dans le cas où il est choisi très grand par rapport au pas d'intégration.

$$\left\| \boldsymbol{\Delta}_{n+1}^{[b]} \right\| = \mathcal{O}(h^p) + \mathcal{O}(H)\,, \tag{A.2a}$$

$$\left\| \boldsymbol{Y}^{[b]}(t_n) - \boldsymbol{Y}_n^{[b]} \right\| = \mathcal{O}(h^p) + \mathcal{O}(H)\,. \tag{A.2b}$$

Pour généraliser ces résultats au cas multi-rythmes, il est considéré que chaque sous-modèle $\sum_b'$ (pour $b = 1, \ldots, B$) peut avoir son propre pas de communication $H^{[b]}$ différent des autres (voir la figure A.10). Une idée très simple consiste à dire que l'inégalité peut être transformée par $\mathcal{O}(h^p) + \mathcal{O}\left(H^{[b]}\right)$.

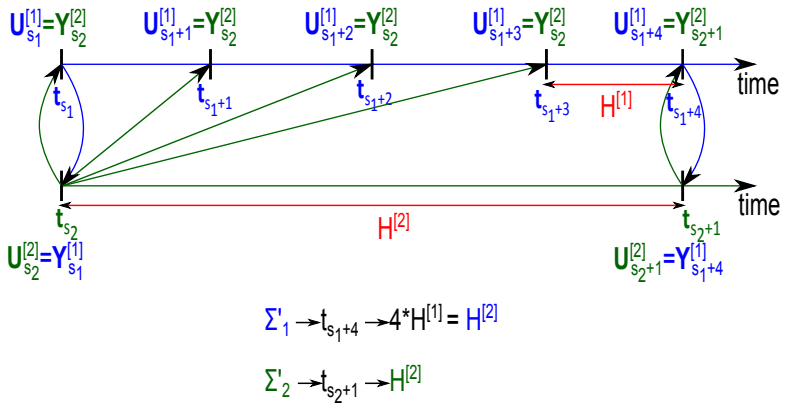

Figure A.10 – Co-simulation multi-rythmes.

Cependant, ce concept ne peut être vrai puisque si nous prenons l'exemple de deux blocs connectés, où l'un est x fois plus rapide que l'autre (voir figure A.10), l'erreur due au pas de communication est en réalité bornée par $\mathcal{O}\left(\max_{b=1,\ldots,Bc} H^{[b]}\right)$ où B_c est l'ensemble des blocs

connectés. Dans cet exemple, les erreurs de $\sum_1'$ et de $\sum_2'$ sont bornées par $\mathcal{O}\left(H^{[2]}\right)$. Pour conclure, la généralisation au cas multi-rythmes consiste à remplacer le terme $\mathcal{O}(H)$ par le terme $\mathcal{O}\left(\max_{b=1,\ldots,Bc} H^{[b]}\right)$.

Pas de communication adaptatif

Nous proposons dans cette partie d'évaluer l'erreur induite due au pas de communication. L'approche proposée est basée sur le principe des solveurs numériques (voir la figure A.11) et elle est destinée à la co-simulation mono-rythme.

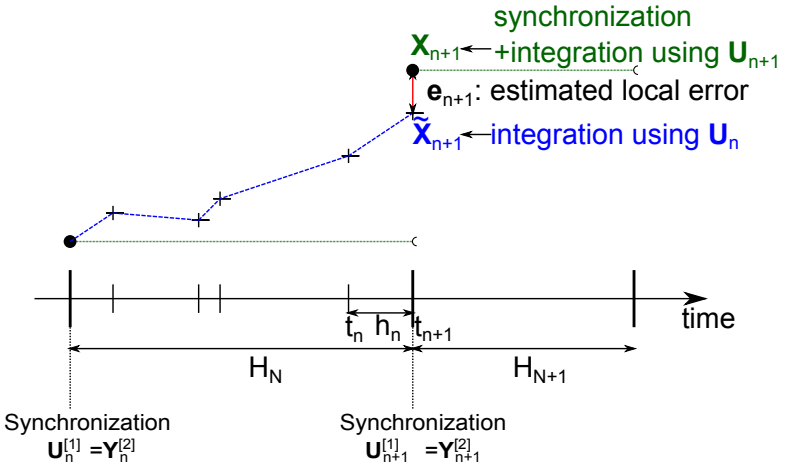

Figure A.11 – Erreur estimée localement pour l'adaptation du pas de communication.

Ce type de variation du pas de communication nécessite que les états puissent faire des "roll-backs", ce qui signifie qu'il faut les sauvegarder pour tous les sous-modèles, à chaque pas de communication. L'indicateur d'erreur $E^{[b]}$,$(i = 1, \ldots, B))$ est calculé comme suit :

$$E_{n+1}^{[b]} = \sqrt{\frac{1}{n_{X^{[b]}}} \sum_{i=1}^{n_{X^{[b]}}} \left(\frac{e_{i,n+1}^{[b]}}{\mathrm{RTol}_i \left| X_{i,n+1}^{[b]} \right| + \mathrm{ATol}_i} \right)^2}, \tag{A.3}$$

où $\boldsymbol{e}_{n+1}$ est le vecteur des erreurs estimées localement $e_{i,n+1}$, calculées à t_{n+1}. Il représente la différence entre les valeurs des états calculées avec le solveur, basées sur $\boldsymbol{U}_n$, et les valeurs des états calculées avec la valeur courante $\boldsymbol{U}_{n+1}$. La solution $\boldsymbol{X}_{n+1}^{[b]}$ est considérée comme suffisamment précise si la condition

$$E_{n+1}^{[b]} \leq 1 \tag{A.4}$$

est vraie.

Lorsqu'au moins un indicateur d'erreur $E^{[b]}$ est supérieur à 1, un indicateur ultime E_{n+1} est calculé comme suit :

$$E_{n+1} = \max_{b=1,\ldots,B} E_{n+1}^{[b]},$$

et le prochain pas de communication est réduit

$$H_{n+1} = \alpha_s \frac{H_n}{E_{n+1}}, \tag{A.5}$$

avec $\alpha_s \in [0.8, 0.9]$ un facteur de sécurité. Cette réduction est effectuée jusqu'à ce que la condition de (A.4) est satisfaite. Dans le cas contraire, pour agrandir le pas de communication suivant (A.5), tous les modèles doivent satisfaire (A.4).

Cette approche est une première proposition qui n'est pas encore testée. En effet, comme il a été mentionné plus tôt, l'adaptation du pas de communication nécessite le "roll-back" des états. Cette capacité n'est pas possible avec la spécification "FMI for Model Exchange 1.0", qui est actuellement implémentée dans xMOD.

Conclusion

L'analyse de convergence de la co-simulation modulaire montre que l'erreur globale des états et des sorties est étroitement liée au couplage entre les modèles, au solveur numérique (ordre, pas de temps) et en particulier au pas de communication.

A.2.4 Décomposition de modèle basée sur une analyse structurelle

Introduction

Souvent, les matrices d'incidence entre les variables d'état, ou entre les variables d'état et les événements, ont une forme creuse. En effet, les événements sont seulement déclenchés par l'évolution d'un sous-ensemble du vecteur d'état, et les discontinuités correspondantes agissent uniquement sur un sous-ensemble du système. Ainsi, pour améliorer la vitesse de simulation, nous proposons de diviser le modèle en sous-systèmes de sorte que le traitement de chaque discontinuité peut être, autant que possible, encapsulé dans un seul sous-système. Le but est d'optimiser l'exploitation du parallélisme des sous-systèmes tout en minimisant l'erreur de retard qui est due au découplage.

Description de la méthode

Afin de minimiser les erreurs de découplage, l'idée est d'analyser les variables d'état afin de réduire les variables couplées entre les sous-systèmes, et ensuite de s'intéresser aux événements afin de proposer une approche qui permet de réduire le nombre d'interruptions au cours de l'intégration en parallélisant la gestion des événements au niveau du solveur.

Deux méthodes de diagonalisation par simple permutation ainsi que leurs outils logiciels associés ont été évalués pour en retenir à la fin une méthode (voir détails dans le chapitre 10). Il s'agit de la modélisation par hypergraphe utilisée par l'outil PaToH [107]. La méthode consiste à transformer une matrice $\boldsymbol{A}$ en un hypergraphe $H = (U, N)$, défini par un ensemble de noeuds (sommets) U et un ensemble de mailles (hyper- arêtes) N entre ces sommets. Cet hypergraphe est utilisé par un PaToH pour le partitionner et obtenir par la suite une matrice $\boldsymbol{A}_{SB}$ bloc-diagonale à simple bord (c'est-à-dire qu'il y a des éléments non-nuls sur les dernières lignes seulement), comme dans la figure A.12.

PaToH est un outil de partitionnement multi-niveaux d'hypergraphes qui se fait en 3 phases : un dé-raffinement, un partitionnement initial et un raffinement. Dans la première phase, un regroupement multi-niveaux est appliqué à l'hypergraphe d'origine qui permet de fusionner les sommets qui sont fortement en interaction pour faire des super-noeuds. Le regroupement se fait à l'aide d'heuristiques et s'arrête au moment où le nombre de sommets est inférieur à

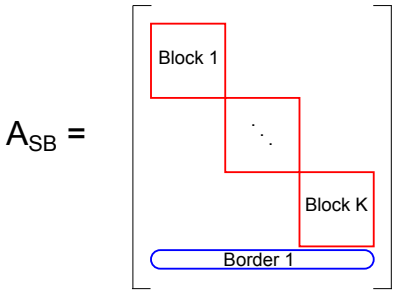

Figure A.12 – Matrice bloc-diagonale avec un simple bord.

un seuil prédéfini. Ensuite, la deuxième phase correspond à la partition de cet hypergraphe grossier utilisant diverses heuristiques. Enfin, dans la troisième phase, la partition obtenue est projetée vers l'hypergraphe original en affinant les partitions projetées à l'aide de différentes heuristiques.

La représentation de la structure non-nulle de la matrice par un modèle d'hypergraphe réduit le problème de permutation à un problème de partitionnement d'hypergraphe. L'hypergraphe correspondant à la matrice $\boldsymbol{A}$ est construit par PaToH en remplaçant les lignes et les colonnes de la matrice par des mailles et des noeuds respectivement. Le nombre d'arcs est égal au nombre d'éléments non-nuls de la matrice. Après la transformation, PaToH partitionne l'hypergraphe comme indiqué dans la figure A.13, en ayant comme objectif de minimiser le nombre de lignes de couplage et d'équilibrer les sous-hypergraphes c'est-à-dire les sous-matrices diagonales.

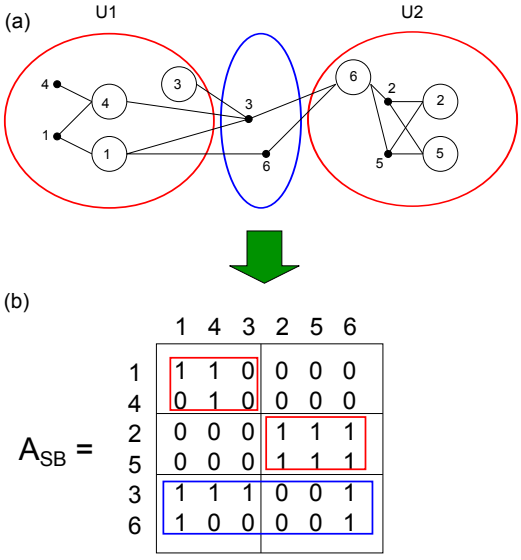

Figure A.13 – (a) Représentation de la matrice $\boldsymbol{A}$ en hypergraphe avec un partitionnement en 2 ; (b) matrice bloc-diagonale $\boldsymbol{A}_{SB}$.

Dans les essais suivants, les relations entre les variables d'états et les événements ainsi que leurs comportements sont essentiels pour l'étude du partitionnement du système. Pour ce faire, le modèle, qui est écrit à l'origine en langage Modelica, doit être traduit en un autre langage plus simple appelé Micro-Modelica (μ-Modelica) [109] pour le rendre compréhensible par l'outil QSS [56] (voir la figure A.14).

Du fait qu'il n'y a pas encore de traducteur automatisé, cette traduction est réalisée manuellement. Ajoutant à cette limitation la complexité et la taille du modèle moteur à quatre cylindres, l'étude a été réduite à un modèle moteur mono-cylindre. Même avec cette restriction de modèle, la traduction reste très longue à faire, de l'ordre du millier en lignes de code et nécessite en plus

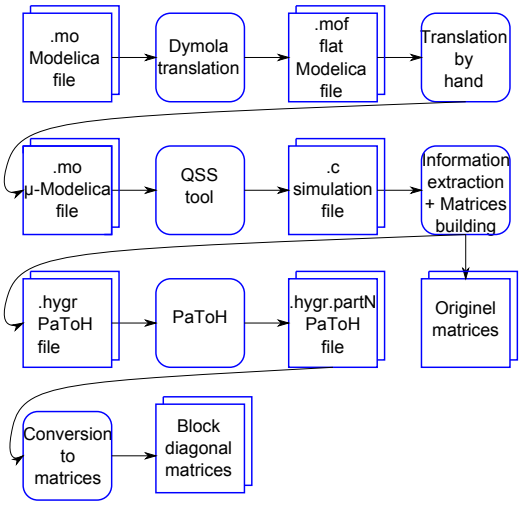

Figure A.14 – Chaine d'outils logicielle.

des connaissances préalables et des astuces pour le débogage (l'outil n'est pas encore mature). Le solveur de l'outil QSS n'est pas utilisé ici, seulement une partie de la chaîne d'outils est utilisée pour générer un fichier appelé fichier de simulation qui contient des informations importantes sur le système et sur les relations entre les états et les événements. Ces données sont extraites par la suite par un outil dédié personnalisé, pour être transformées sous une forme de matrice pour la visualisation et sous une forme de fichier hypergraphe pour l'outil de PaToH. Enfin, PaToH génère le fichier hypergraphe partitionné qui décrit la façon dont le graphe est décomposé et qui est transformé par la suite sous forme de matrice pour la visualisation.

Validation expérimentale

Le cas d'étude étant un modèle moteur mono-cylindre qui est caractérisé par 15 variables d'état $\boldsymbol{X}$, 111 événements $\boldsymbol{Z}$ et 93 variables discrètes $\boldsymbol{D}$. Les événements z_i $(i = 1, \ldots, n_Z)$ et les variables discrètes d_i $(i = 1, \ldots, n_D)$ sont définis dans les blocs "when" comme suit :

```
when (z_i) then
 d_i = ... ;
elsewhen !(z_i) then
 d_i = ... ;
end when;
```

En construisant la matrice d'incidence des états (figure A.15), on constate que parmi les 15 variables d'état, il y a seulement 6 qui sont directement couplées. Plus précisément, c'est le calcul de $\dot{X}_{13}$ qui dépend des valeurs X_0, X_{10}, X_{11}, X_{12} et X_{14}. Si l'analyse s'arrête à cette relation, seulement 40% des variables d'état sont considérées comme directement couplées, tandis que les autres ne dépendent que d'entrées externes et peuvent même rester constantes pour des cas d'application particuliers (par exemple, quand la vitesse du moteur est constante).

Le même constat est fait pour les événements puisque seulement 39 événements parmi les 111 sont activés par les variables d'état comme c'est illustré dans la figure A.16.

Ce nombre représente seulement 35% du nombre total d'événements, alors que le reste est juste utilisé pour activer d'autres événements. En fait, les 72 événements restants sont définis dans la bibliothèque ModEngine pour être utilisés dans d'autres systèmes plus complexes, et non pour l'utilisation du cas particulier du mono-cylindre. En conséquence, seul le sous-ensemble des événements actifs devrait être détecté.

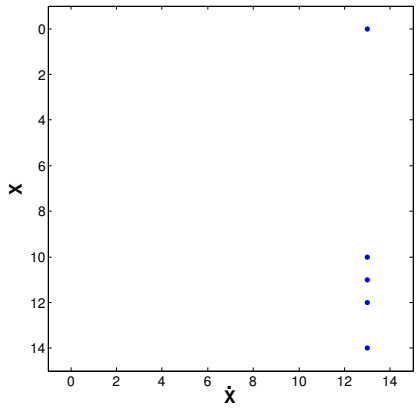

Figure A.15 – Matrice d'incidence des états : $\dot{\boldsymbol{X}}$ en fonction de $\boldsymbol{X}$.

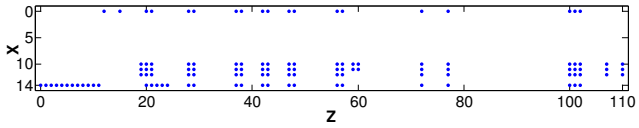

Figure A.16 – Matrice d'incidence des événements : $\boldsymbol{Z}$ en fonction de $\boldsymbol{X}$.

Toutefois, si les variables d'état $\boldsymbol{X}$ peuvent affecter les événements $\boldsymbol{Z}$, les événements, à leur tour, peuvent également modifier les valeurs des variables d'état. Pour construire la matrice d'incidence correspondante, il faut à la fois construire la matrice d'incidence définissant les variables discrètes $\boldsymbol{D}$ influencées par les événements $\boldsymbol{Z}$: $\boldsymbol{Z} \rightarrow \boldsymbol{D}$ (voir figure A.17) et la matrice d'incidence définissant les dérivées des états $\dot{\boldsymbol{X}}$ influencées par les variables discrètes $\boldsymbol{D}$: $\boldsymbol{D} \rightarrow \dot{\boldsymbol{X}}$ (voir figure A.18).

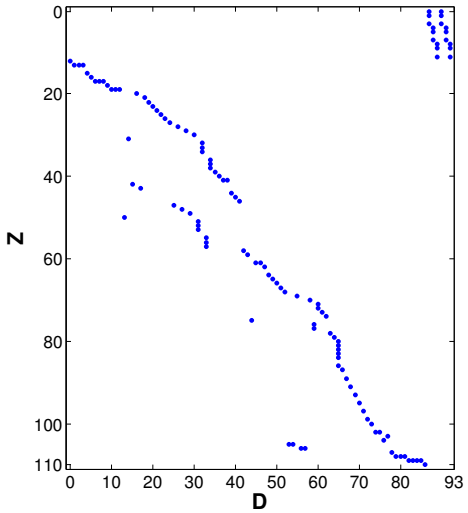

Figure A.17 – Matrice d'incidence : $\boldsymbol{D}$ influencées par $\boldsymbol{Z}$.

Ainsi, la matrice d'incidence $\boldsymbol{Z} \rightarrow \dot{\boldsymbol{X}}$ est déduite par transitivité dans la figure A.19. Elle montre que l'identification préalable de certaines variables d'état comme étant non couplées (basée sur la matrice d'incidence des états A.15) et celle des événements/états A.16, n'est pas complète. En effet, 13 variables d'état parmi 15 apparaissent dans cette nouvelle matrice

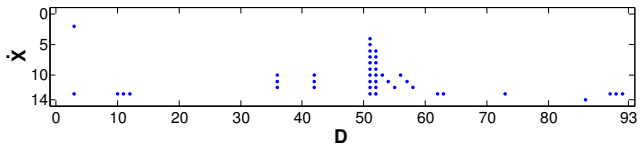

Figure A.18 – Matrice d'incidence : $\dot{\boldsymbol{X}}$ influencées par $\boldsymbol{D}$.

d'incidence. Seulement les variables d'état X_1 et X_3 n'y figurent pas parce qu'elles sont inhibées momentanément pour tester un scénario particulier.

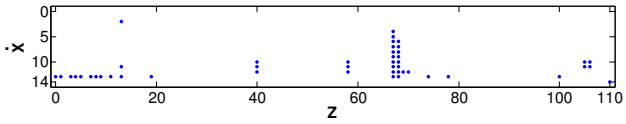

Figure A.19 – Matrice d'incidence : $\dot{\boldsymbol{X}}$ influencées par $\boldsymbol{Z}$.

La combinaison des deux matrices A.16 et A.19, permet de construire la matrice d'incidence des événements/états illustrée dans la figure A.20.

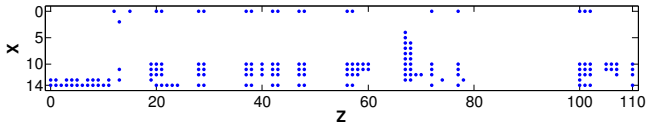

Figure A.20 – Matrice d'incidence : échange de données entre $\boldsymbol{Z}$ et $\boldsymbol{X}$.

Une fois les états et les événements inutiles éliminés, l'imbrication dans les deux sens de dépendance des états et des événements restants montre qu'il est difficile de partitionner le système. Par ailleurs, la matrice d'incidence des états peut être reconstruite différemment que A.15, à partir des matrices A.16 ($\boldsymbol{X} \rightarrow \boldsymbol{Z}$) et A.19 ($\boldsymbol{Z} \rightarrow \dot{\boldsymbol{X}}$), en passant par les événements. Le résultat est illustré dans la figure A.21.

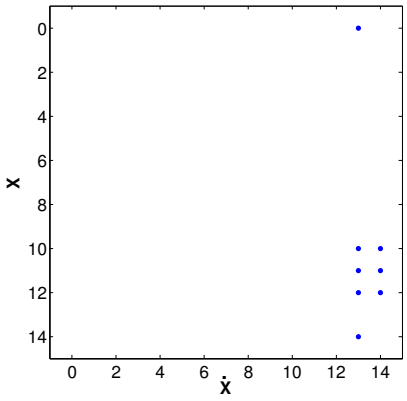

Figure A.21 – Matrice d'incidence des états : $\dot{\boldsymbol{X}}$ influencées par $\boldsymbol{X}$.

Cette nouvelle manière de construction (à travers les événements $\boldsymbol{Z}$), montre que la dérivée $\dot{X}_{14}$ est également en fonction de X_{10}, X_{11} et X_{12}. Par conséquent, afin de déterminer correctement les relations entre les variables, il est important d'utiliser toutes les données disponibles du système, directement et par transitivité.

Enfin, les résultats sur les relations entre les dérivées, les états et les événements montrent que le mono-cylindre est modélisé d'une manière séquentielle ce qui rend difficile, voire impossible, la parallélisation de l'exécution du modèle. Cette découverte confirme le positionnement des spécialistes de moteurs pour ne pas séparer la chambre de combustion du vilebrequin par exemple, étant donné qu'ils partagent les mêmes données aux mêmes instants.

Une alternative à ce résultat est d'analyser la relation entre les événements pour étudier la possibilité de séparer certains événements en blocs au niveau du solveur. Cette décomposition pourrait faciliter la détection et la localisation des événements. Dans cet objectif, la matrice d'incidence des événements $\boldsymbol{Z}(ligne) \rightarrow \boldsymbol{Z}(colonne)$ est construite par transitivité (voir figure A.22), en utilisant les matrices d'incidence qui définissent $\boldsymbol{Z} \rightarrow \boldsymbol{D}$ (voir figure A.17) et $\boldsymbol{D} \rightarrow \boldsymbol{Z}$.

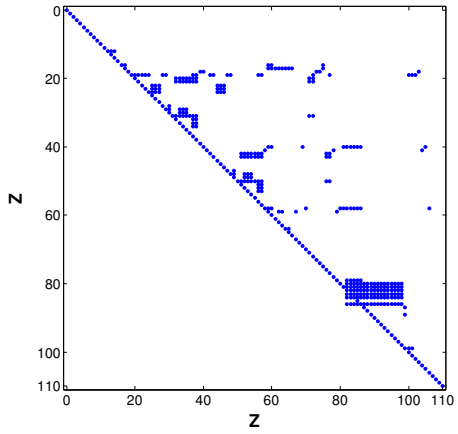

Figure A.22 – Matrice d'incidence des événements.

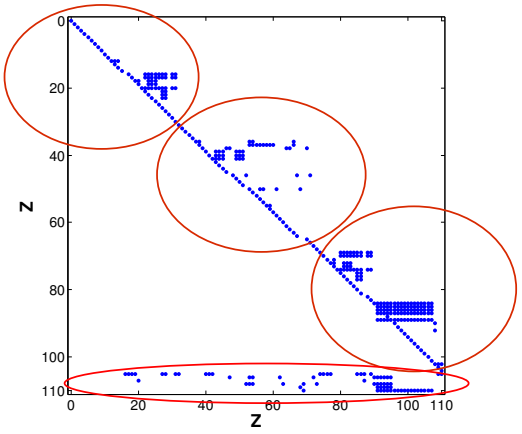

Figure A.23 – Matrice d'incidence des événements bloc-diagonale.

A l'inverse des matrices basées sur les états, la matrice d'incidence des événements peut être transformée en une matrice bloc-diagonale de 3 blocs avec l'outil PaToH. Chaque bloc peut donc être considéré comme un sous-système où toutes les discontinuités directement liées appartiennent à la même entité (voir la figure A.23).

Ces blocs peuvent être parallélisés pour espérer que le temps d'exécution soit réduit. En fait, la détection et la localisation des événements avec le redémarrage du solveur augmente considérablement le temps d'intégration (voir la figure A.24). En bref, pour l'intégration du mono-cylindre par LSODAR, le temps de calcul est en moyenne 4 fois plus long quand il y a des événements à traiter et peut aller jusqu'à 60 fois le temps normal. Ceci confirme l'intérêt de la bloc-diagonalisation pour limiter le nombre d'interruptions par bloc et la parallélisation des traitements des événements au niveau du solveur. Cependant, cela ne pouvait pas être expérimentalement testé en raison de l'indisponibilité actuelle de la librairie d'exécution.

Conclusion

Ces méthodes de décomposition basées sur des analyses structurelles des modèles permettent d'aider l'utilisateur sur la façon de partitionner un grand système. Le cas d'étude particulier du mono-cylindre montre la difficulté du problème de décomposition du système, que ce soit d'un point de vue physique, ou à partir des relations entre états et événements. En effet, lorsque la matrice d'incidence n'est pas creuse, il n'est pas possible de découpler le système en une forme bloc-diagonale. Cependant, la matrice d'incidence des événements est généralement creuse, donc

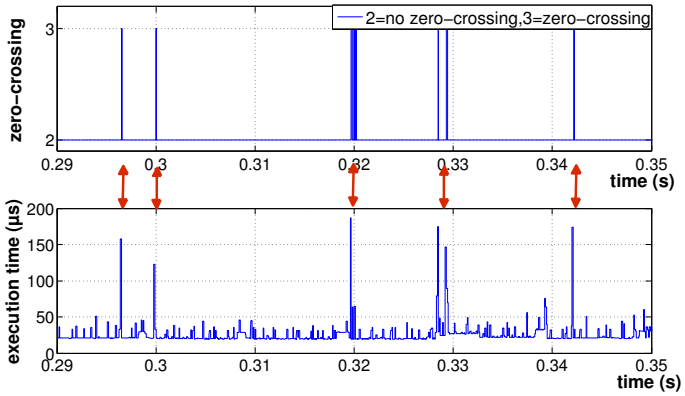

Figure A.24 – Effet de la gestion des événements sur le temps d'exécution.

sa bloc-diagonalisation est plus facilement faisable. Ainsi, une manière pertinente de paralléliser serait d'exécuter en parallèle la gestion des événements par le solveur.

A.2.5 Co-simulation avec ordonnancement à grains fins : RCosim

Introduction

A partir d'un modèle complexe déjà partitionné (suivant les méthodes décrites précédemment) en plusieurs modèles de complexité moindre, même avec un ordre d'exécution efficace dans l'approche de co-simulation modulaire, quand tous les modèles sont DF la co-simulation modulaire entraîne toujours des sorties décalées ou retardées (voir la section 9.3.2). Pour tirer profit du partitionnement sans pour autant ajouter des retards inutiles, il est judicieux d'aborder le problème avec un point de vue ordonnancement à grains fins. Grâce à la spécification FMI, il est possible d'accéder à des informations concernant la dépendance interne entre Entrées/Sorties d'un modèle encapsulé dans un FMU (voir figure A.25).

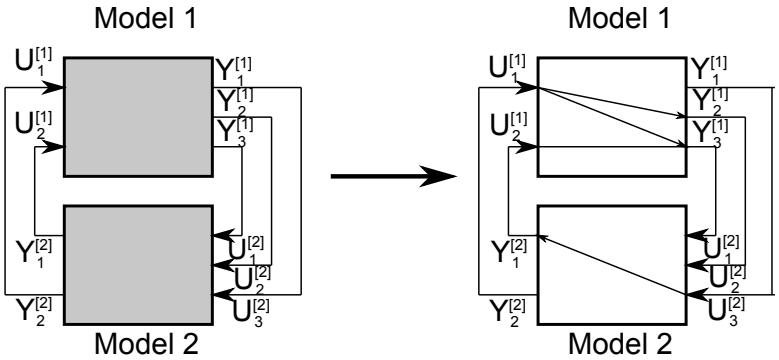

Figure A.25 – Connexion Entrées/Sorties à partir une vue intra and inter modèles.

Par conséquent, le traitement de la co-simulation peut être affiné. Au lieu de considérer l'ensemble du module comme DF ou NDF, il est possible avec la spécification FMI de classifier les sorties en les identifiant localement en tant que DF ou NDF. Par exemple, dans la figure A.25, le modèle 1 et le modèle 2 sont tous les deux considérés DF à l'échelle du module. L'exploration des liens entre les Entrées/Sorties à l'intérieur de chaque modèle révèle qu'il n'y a pas de cycle qui ne contienne que des sorties DF. De plus, un FMU fournit différentes fonctions pour calculer

chaque sortie séparément, et une en particulier pour mettre à jour (intégrer) les états du modèle. Par la connaissance des deux dépendances intra et inter modèles entre les Entrées/Sorties, ces fonctions permettent différentes possibilités d'exécution sans un ordre strict d'exécution des modèles. La granularité de parallélisation est donc augmentée et la répartition des différentes opérations entre les différents processeurs devient un problème plus complexe.

Description de la méthode

Une co-simulation de différentes FMUs, avec des pas de communication constants, peut être décrite par un graphe orienté où les sommets sont les opérations et les arcs sont des relations de précédence entre ces opérations. De plus, sachant que le modèle global est décrit par des équations différentielles ordinaires et ne présente pas de boucles algébriques, ce graphe est donc nécessairement un graphe acyclique direct (voir la figure A.26). Plus précisément, les opérations sont des appels de fonctions de mise à jour de sortie ($update_{\text{out}}$), d'entrée ($update_{\text{in}}$) ou du vecteur d'état ($update_{\text{state}}$). Un arc d'une opération $update_{\text{out}}$ à une opération $update_{\text{in}}$ correspond à une dépendance des données inter-modèles (par exemple de $Y_2^{[2]}$ à $U_1^{[1]}$ dans la figure A.26). Ces arcs expriment les dépendances de données entre les modèles. Un arc d'une opération $update_{\text{in}}$ à une opération $update_{\text{out}}$ exprime une dépendance DF intra-modèle (par exemple de $U_3^{[2]}$ à $Y_1^{[2]}$ dans la figure A.26). Ces dépendances sont répertoriées dans chaque modèle FMU. Il y a aussi un arc de chaque $update_{\text{in}}$ à $update_{\text{state}}$ du même modèle (par exemple de $U_2^{[1]}$ à $\dot{\boldsymbol{X}}^{[1]}$ dans la figure A.26), qui signifie que toutes les entrées du modèle sont nécessaires pour mettre à jour l'état du modèle. Enfin, il y a un arc de chaque $update_{\text{out}}$ à $update_{\text{state}}$ du même modèle (par exemple de $Y_3^{[1]}$ à $\dot{\boldsymbol{X}}^{[1]}$ dans la figure A.26), parce que le calcul de $\boldsymbol{Y}_k$ a besoin de la valeur $\boldsymbol{X}_k$, qui n'est plus disponible après le calcul de $update_{\text{state}}$ qui donne $\boldsymbol{X}_{k+1}$. Pour exécuter une co-simulation, l'exécution de la totalité du graphe est requise à chaque pas de communication. Néanmoins, il faut que la précédente exécution du graphe soit entièrement terminée avant de commencer la nouvelle.

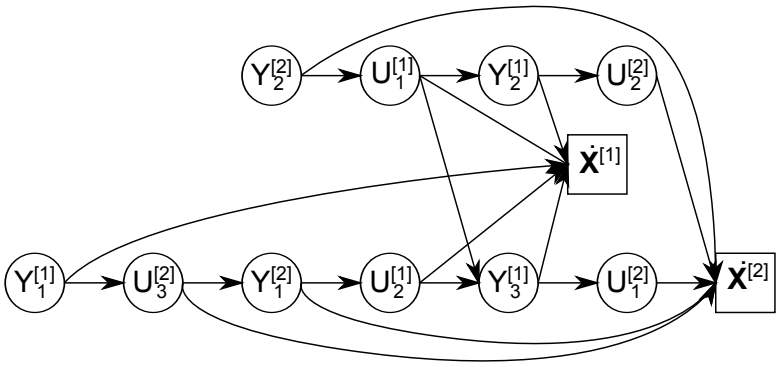

Figure A.26 – Graphe de dépendances à grains fins.

Pour que la simulation multi-coeurs soit rapide, les opérations doivent être distribuées et ordonnancées sur les différents coeurs disponibles. Pour être efficace, l'ordonnancement distribué doit prendre en compte le temps de calcul de chaque opération. Nous proposons d'utiliser une approche d'heuristique hors ligne, similaire à celle de [22], qui cherche à optimiser la répartition et l'ordonnancement des différentes opérations des modèles sur les différents coeurs disponibles.

L'heuristique considère les dates de début et de fin pour chaque opération pour tendre à minimiser la longueur de chemin critique d'un graphe orienté acyclique, dans lequel une durée d'exécution C_i est rattaché à chaque opération OP_i. Pour des fins de simulation temps-réel, ces

durées sont estimées par leur WCET. Ici, l'objectif est de réaliser une simulation rapide, qui n'est pas critique, ainsi ces durées peuvent être estimées (par exemple à partir d'une simulation mono-coeur de référence). Dans les exemples pratiques, les opérations $update_{\text{state}}$ sont de loin plus coûteuses que les opérations $update_{\text{out}}$ (soit $C(update_{\text{state}}) \gg C(update_{\text{out}})$) , tandis que les opérations $update_{\text{in}}$ sont de simples copies de données dont le coût est négligeable (c'est-à-dire $C(update_{\text{in}}) \ll \varepsilon$).

L'heuristique de la fonction coût calcule la pression d'ordonnancement d'une opération donnée sur un coeur spécifique. Cette pression d'ordonnancement est la différence entre l'augmentation du chemin critique (en définissant cette opération sur ce coeur) et la flexibilité de l'opération (différence entre la date de début au plus tôt et la date de fin au plus tard). A chaque étape, pour chaque opération restante dont tous les prédécesseurs ont déjà été ordonnancés, l'heuristique calcule sa pression d'ordonnancement sur chaque coeur, et l'affecte au meilleur coeur, c'est-à-dire celui qui minimise la pression. Ensuite, parmi toutes ces opérations en attente, celle qui a la plus grande pression (sur son meilleur coeur) est sélectionnée et ajoutée à l'ordonnancement.

Néanmoins, il existe d'autres contraintes d'allocation pour les opérations. En effet, la spécification FMI ne force pas une opération FMU à être "thread safe" et actuellement les opérations du FMU $update_{\text{out}}$ ne peuvent pas s'exécuter en parallèle. Puisque cette contrainte pourrait être résolue dans le futur, il a été décidé de réduire temporairement l'espace de recherche de l'heuristique. Toutes les opérations appartenant au même FMU sont affectées au même coeur, qui a été choisi par l'heuristique pour la première opération. Chaque fois qu'une opération est ordonnancée, des opérations de synchronisation sont insérées si c'est nécessaire. Par exemple, si les opérations $Y_1^{[1]}$ et $U_3^{[2]}$ sont allouées par l'heuristique sur différents coeurs, un sémaphore est signalé juste après $Y_1^{[1]}$ sur le même coeur, et un sémaphore d'attente est exécuté sur l'autre coeur juste avant $U_3^{[2]}$.

Comparée à des approches de co-simulation distribuée avec une granularité au niveau modèle, cette approche raffinée présente deux avantages importants. D'une part, l'utilisation d'une granularité plus fine augmente potentiellement les possibilités de découplage des modèles et permet d'augmenter l'accélération de la co-simulation. D'autre part, les dépendances entre les entrées et les sorties de modèles sont satisfaites à travers les dépendances intra et inter modèles, permettant ainsi de trouver un ordonnancement valide sans l'insertion de retards inutiles. Cela rend les résultats de co-simulation plus proche à ceux de la référence.

Validation expérimentale

Les tests sont réalisés sur une plate-forme 16 GB de RAM et 2 processeurs "Intel Xeon" à 8 coeurs dont chacun s'exécute à 3.1 GHz. Par ailleurs, le modèle du moteur CFM partitionné, décrit précédemment, détient 91 entrées et 98 sorties et il est simulé dans XMOD suivant l'approche de co-simulation à granularité fine appelée " RCosim" qui couvre l'ordonnancement de 103 opérations (5 $update_{\text{state}}$ and 98 $update_{\text{out}}$).

Cette étude compare les performances de simulation (vitesse et précision de simulation), entre les deux approches précédentes de co-simulation modulaire " MCosim " et la co-simulation à grains fins "RCosim". Pour le cas d'étude, toutes les entrées et les sorties sont mises à jour en suivant l'ordre d'ordonnancement de l'heuristique, puis l'intégration des modèles de la boucle d'air (AP) et des cylindres est réalisée en parallèle.

L'objectif de l'approche "RCosim" est d'améliorer, par rapport à la méthode "sv-MCosim", un peu la précision des résultats et de réduire énormément le temps de simulation. De plus, compa-

rée à la méthode “ev-MCosim”, son but est d’améliorer énormément la précision des résultats au prix d’une potentielle faible augmentation du temps de simulation. Le terme “potentielle” est utilisé parce que cette augmentation peut être équilibrée ou même éliminée en utilisant un solveur basé sur un contrôle d’erreur.

La figure A.27 rappelle et résume les différentes méthodes. DT est le temps d’exécution pendant un pas de communication et il englobe l’intégration des modèles dans les blocs X_i (par exemple X_{AP} pour AP) ainsi que les mises à jour d’Entrées/Sorties, dans les blocs IN/OUT pour “MCosim” et dans les blocs IN/OUT/WAIT pour “RCosim” (les temps d’attente sont introduits par l’heuristique d’ordonnancement). Chaque modèle est simulé sur un coeur distinct avec son propre solveur.

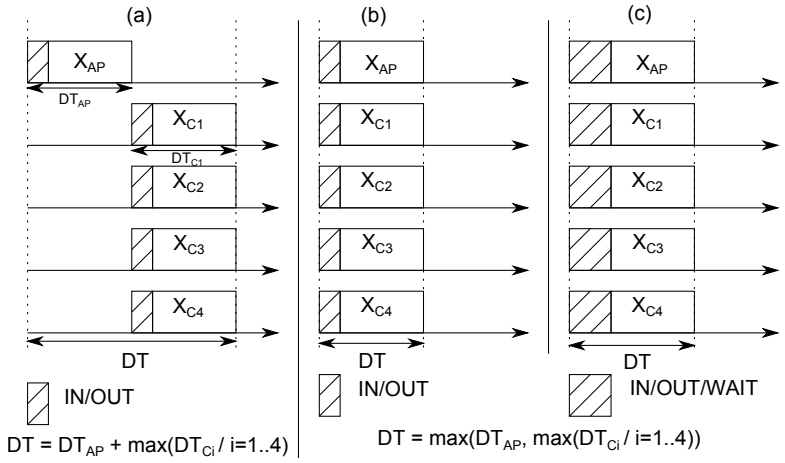

Figure A.27 – (a) méthode sv-MCosim ; (b) méthode ev-MCosim ; (c) méthode RCosim.

La validation du modèle est basé sur l’observation de certaines quantités d’intérêt comme les pression d’admission et d’échappement du collecteur, respectivement “Pman” et “Pexh”, l’AFR et le couple. Ces sorties sont calculées en utilisant LSODAR. La simulation de référence Y_{ref} est construite à partir de l’intégration du modèle moteur entier, la tolérance du solveur (Tol) a été diminuée jusqu’à atteindre des résultats stables, pour $\mathrm{Tol} = 10^{-7}$ (au prix d’une vitesse de simulation inacceptable). Ensuite, pour explorer le compromis entre vitesse et précision, des itérations de simulation ont été réalisées en augmentant la tolérance pour trouver la valeur maximale qui permet d’avoir une précision désirée définie par $Er \leq 1\,\%$ (voir le tableau A.2), où Er étant l’erreur d’intégration relative Er définie dans (A.6). Cette tolérance correspond à $\mathrm{Tol} = 10^{-4}$.

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

avec N le nombre de points sauvegardés durant 1 s de simulation.

Table A.2 – Erreur d’intégration relative.

Sorties	Pman	Pexh	Couple	AFR
$Er(\%)$	0.027	0.05	0.38	0.37

La co-simulation modulaire exécute pour chaque modèle toutes les opérations $update_{\mathrm{out}}$ en un seul bloc comme pour l’opération $update_{\mathrm{state}}$. La figure A.28 illustre le chronogramme et montre la période d’attente du calcul de AP qui représente la différence entre “ sv-MCosim” et “ ev-MCosim”.

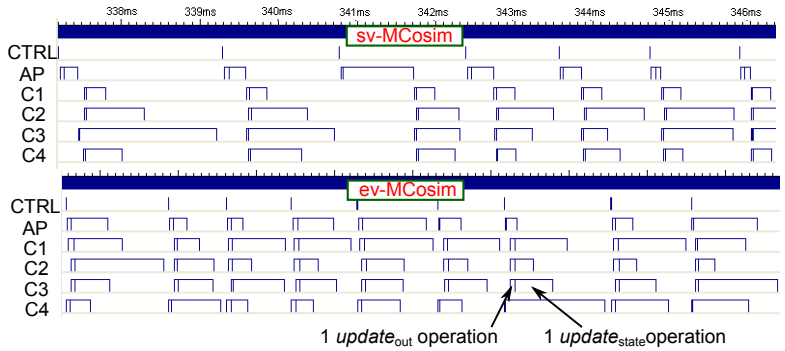

Figure A.28 – Chronogramme de MCosim.

La co-simulation "RCosim" ordonnance les 103 opérations de $update_{\text{out}}$ et de $update_{\text{state}}$. Comme décrit dans la figure A.29, les temps de calcul de $update_{\text{out}}$ sont négligeables par rapport à $update_{\text{state}}$. Un zoom sur le chronogramme montre l'ordonnancement des opérations $update_{\text{out}}$.

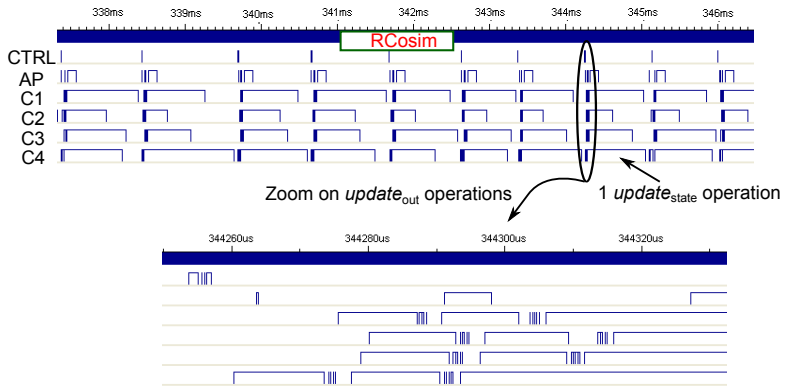

Figure A.29 – Ordonnancement des opérations de mise à jour.

Le couple est une sortie DF du AP car il est la somme des quatre couples fournis par les cylindres. Comme prévu, les résultats montrent que le couple est retardé d'un pas de communication avec les approches "MCosim", étant donné que tous les modèles sont DF. Cependant, grâce à la méthode "RCosim", le couple est presque identique à la référence, comme c'est indiqué dans la figure A.30.

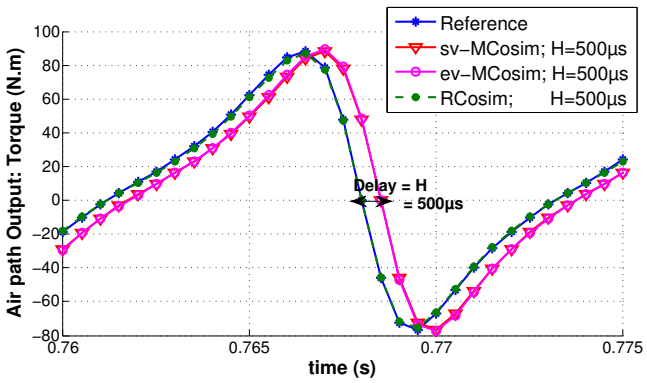

Figure A.30 – Comportement d'une sortie DF suivant les différentes méthodes.

L'erreur d'intégration relative est calculée, par la suite, pour plusieurs pas de communication

(voir le tableau A.3). Les résultats montrent que la méthode “RCosim” maintient la stabilité de l'intégration, même pour de grands pas de communication. En effet, Er reste proche de 1 %, alors que les méthodes “MCosim” souffrent d'importantes erreurs dues au retard, de l'ordre de 20 %.

Table A.3 – Erreur d'intégration relative du couple.

Méthode de simulation	sv-MCosim	ev-MCosim	RCosim
Er(%) avec $H = 100\,\mu s$	2.95	4.38	0.68
Er(%) avec $H = 250\,\mu s$	9.12	9.33	1.1
Er(%) avec $H = 500\,\mu s$	19.83	19.19	1.37

La pression d'admission est une sortie NDF de AP. Dans ce cas, il n'y a pas de retard quelle que soit la méthode. Comme pour le couple, l'erreur d'intégration relative de la pression dépend également du pas de communication (voir la figure 11.7). Cependant, la largeur du pas n'est plus aussi néfaste car il n'y a pas de problème de boucle.

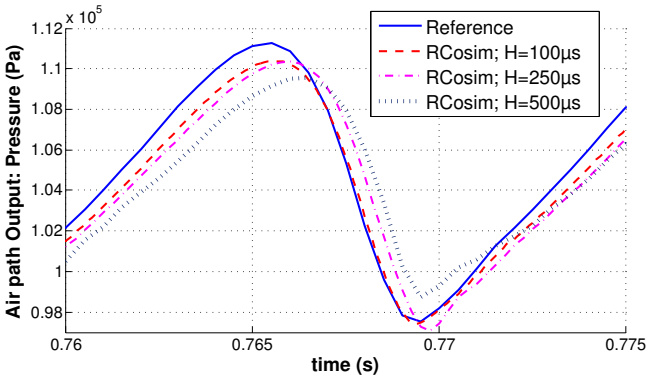

Figure A.31 – Effet du pas de communication sur une sortie NDF avec RCosim.

Le tableau A.4 montre que la méthode “RCosim” est encore avantageuse pour la précision de la simulation. Pour atteindre l'erreur souhaitée à la fois pour les sorties DF et NDF, le pas de communication doit être limité à 100 µs pour les co-simulation modulaire “MCosim” alors qu'il peut être agrandi jusqu'à 500 µs avec la co-simulation à granularité fine “RCosim”.

Table A.4 – Erreur d'intégration relative de la pression d'admission.

Méthode de simulation	sv-MCosim	ev-MCosim	RCosim
Er(%) avec $H = 100\,\mu s$	0.61	0.63	0.5
Er(%) avec $H = 250\,\mu s$	1.2	1.11	0.88
Er(%) avec $H = 500\,\mu s$	1.8	1.75	1.23

L'intégration du modèle moteur (118 de variables d'état et 312 d'événements) est très coûteuse en temps. Avec $\text{Tol} = 10^{-4}$, le temps de simulation du moteur non partitionné est 76.5 fois plus lent que le temps réel. En comparaison avec la référence, les accélérations ont été mesurées pour $H = 250\,\mu s$ (afin de garder $Er \approx 1\,\%$). Le tableau A.5 montre que l'accélération atteint 7.82 pour “sv-MCosim” et 8.84 pour “ev-MCosim”. La plus grande accélération est obtenue grâce à “RCosim” qui atteint 10.87, de sorte que la vitesse de simulation est maintenant seulement 7.04 fois plus lente que le temps réel. En effet, en intégrant avec les bonnes (non retardées) valeurs des entrées, le solveur à pas de temps variable trouve plus rapidement le pas d'intégration le

plus large possible, tout en respectant $Er \approx 1\,\%$. Cette accélération ne peut pas être observée avec un solveur à pas fixe.

Table A.5 – Accélération de la simulation avec les différentes approches.

Méthode de simulation	sv-MCosim	ev-MCosim	RCosim
Accélération (5 coeurs)	7.82	8.84	10.87

Conclusion

La technique " RCosim" conserve l'avantage de la co-simulation modulaire, c'est-à-dire l'accélération supra-linéaire de la simulation. Par ailleurs, " RCosim" améliore la précision des résultats de simulation à travers une heuristique d'ordonnancement hors-ligne des opérations Entrées/Sorties des modèles.

A.2.6 Extrapolation basée sur le contexte

Introduction

Compte tenu du compromis entre la précision des résultats de simulation qui requiert une synchronisation assez fréquente des sous-modèles couplés et l'accélération du temps de simulation qui nécessite de larges pas de synchronisation, nous proposons pour améliorer ce compromis, d'extrapoler les entrées des sous-modèles afin de compenser l'élargissement des pas de communication. En effet, il a été prouvé dans le chapitre 9 que la précision des solutions numériques de la co-simulation modulaire, est de 1^{er} ordre, $\mathcal{O}(H)$, quand le pas de communication H est choisi très grand. En effet, en considérant les entrées des sous-modèles constantes entre deux intervalles de synchronisation, cela joue le rôle d'un bloqueur d'ordre zéro (extrapolation constante). Pour généraliser la borne d'erreur, le terme $\mathcal{O}(H)$ peut être alors remplacé par $\mathcal{O}\left(H^{k+1}\right)$, où k est l'ordre d'extrapolation. En utilisant par exemple une extrapolation linéaire ($k = 1$) ou quadratique ($k = 2$) à la place d'une extrapolation constante ($k = 0$), la borne d'erreur peut être alors réduite.

Description de la méthode

Nous avons choisi de baser notre extrapolation sur les méthodes polynomiales pour leur simplicité et la facilité de mise en oeuvre. En effet, d'après les premiers travaux dans [1, Chapitre 16], la prédiction polynomiale permet des calculs rapides et causaux.

On note u le signal régulièrement échantillonné à chaque pas de communication. $P_{(\delta,\lambda)}$ est le prédicteur polynomial des moindres carrés, de degré $\delta \in \mathbb{N}$ et de longueur de prédiction $\lambda \in \mathbb{N}^*$. La longueur de prédiction λ représente le nombre d'échantillons passés, requis pour chaque prédiction, réalisée suivant la méthode des moindres carrés [115, p.227 sq.]. Pour plus de commodité, nous utilisons une convention d'indice appelée "0-dernier-échantillon" qui consiste à ré-indexer la trame des λ échantillons passés de sorte que le dernier échantillon connu est indexé par 0. Les calculs pour la prédiction à l'instant relatif τ, défini dans (9.19), nécessitent donc les échantillons $\{u_{1-\lambda}, u_{2-\lambda}, \ldots, u_0\}$. Le modèle générique d'extrapolation s'écrit alors sous

la forme de matrice suivante :

$$u(\tau) = \begin{bmatrix} 1 & \tau & \cdots & \tau^\delta \end{bmatrix} \times \begin{bmatrix} \Sigma_\lambda^0 & -\Sigma_\lambda^1 & \cdots & (-1)^\delta \Sigma_\lambda^\delta \\ -\Sigma_\lambda^1 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ (-1)^\delta \Sigma_\lambda^\delta & \cdots & \cdots & \Sigma_\lambda^{2\delta} \end{bmatrix}^{-1} \times \begin{bmatrix} m^\delta \\ -m^{\delta-1} \\ \vdots \\ (-1)^\delta m^0 \end{bmatrix}.$$

avec $m^d = \sum_{l=0}^{\lambda-1} l^{\delta-d} u_{-l}$ et $\Sigma_\lambda^d = \sum_{i=0}^{\lambda-1} i^d$.

Cependant, les systèmes complexes réels présentent généralement des non-linéarités et des discontinuités, de sorte qu'il est difficile de prédire leur comportement futur à partir des observations passées. De plus, les modèles étudiés sont générés en utilisant la spécification "FMI for Model Exchange", qui ne fournit pas les dérivées des signaux d'entrée (à l'inverse du "FMI for Co-Simulation)". Par conséquent, la prédiction polynomiale précédemment décrite ne peut pas extrapoler correctement le long de toutes les trajectoires du système. Par exemple, [105] a étudié une méthode basée sur une implémentation séquentielle de systèmes dynamiques continus qui utilise une extrapolation constante, linéaire ou quadratique et une interpolation linéaire afin d'améliorer la précision de la co-simulation modulaire. L'étude montre que la méthode est efficace pour les systèmes non raides mais qu'elle échoue dès lors que le système devient raide.

Notre but est de définir une méthode destinée à la simulation parallèle des systèmes dynamiques hybrides. Nous empruntons une approche d'extrapolation basée sur le contexte, plus connue dans les codeurs d'images sans pertes [111], tel que GIF ou PNG. Nous adaptons une extrapolation basée sur le contexte qui permet de tenir compte des changements de seuil, de la raideur, des discontinuités ou des comportements assez bizarres, et d'utiliser la prédiction la plus adéquate pour limiter de manière excessive toute prédiction erronée.

Conformément à la précédente convention d'indice "0-dernier-échantillon" et pour des raisons de simplicité, nous définissons d'abord une mesure de la variation basée sur les trois derniers échantillons : $d_0 = u_0 - u_{-1}$ et $d_1 = u_{-1} - u_{-2}$, le dernier et l'avant dernier écarts. Leurs valeurs absolues sont comparées à deux seuils, respectivement γ_0 et γ_{-1}. Nous définissons alors trois conditions complémentaires :

- O si $|d_i| = 0$;
- C_i si $0 < |d_i| \leq \gamma_i$;
- $\overline{C_i}$ si $|d_i| > \gamma_i$.

Par la suite, nous définissons la table des six contextes dans le tableau A.6, ainsi que des exemples pour leurs heuristiques associées aux prédicteurs polynomiaux. Les six contextes sont mutuellement exclusifs et couvrent toutes les options possibles pour un système dynamique hybride. Ils sont illustrés dans la figure A.32. Leurs noms représentent leur comportement. Par exemple, le contexte "flat" traite des signaux constants, pour qui un simple bloqueur d'ordre zéro suffit, donc $P_{(0,1)}$. Le contexte "calm" représente une situation suffisamment échantillonnée, où la valeur des incréments au cours du temps restent en dessous des seuils fixés. Dans ce cas, le signal est relativement régulier, et peut être approché par un polynôme du second ordre, par

Table A.6 – Résumé de la table des six contextes.

n(oms)	#	$\lvert d_{-1}\rvert$	$\lvert d_0\rvert$	$d_{-1}.d_0$	(δ,λ)
f(lat)	0	O	O	O	$(0,1)$
c(alm)	1	C_{-1}	C_0	$\forall$	$(2,5)$
m(ove)	2	C_{-1}	$\overline{C_0}$	$\forall$	$(0,1)$
r(est)	3	$\overline{C_{-1}}$	C_0	$\forall$	$(0,2)$
t(ake)	4	$\overline{C_{-1}}$	$\overline{C_0}$	> 0	$(1,3)$
j(ump)	5	$\overline{C_{-1}}$	$\overline{C_0}$	< 0	$(0,1)$

exemple $P_{(2,5)}$. Pour les contextes “flat” et “jump”, il existe une procédure supplémentaire qui consiste à réinitialiser l'extrapolation pour empêcher une prédiction incorrecte. Par exemple, lorsque le contexte 1 est choisi seulement après le contexte 5, l'extrapolation quadratique $P_{(2,5)}$ doit avoir 5 échantillons valides, alors que les 3 dernières seulement sont pertinentes. Notre

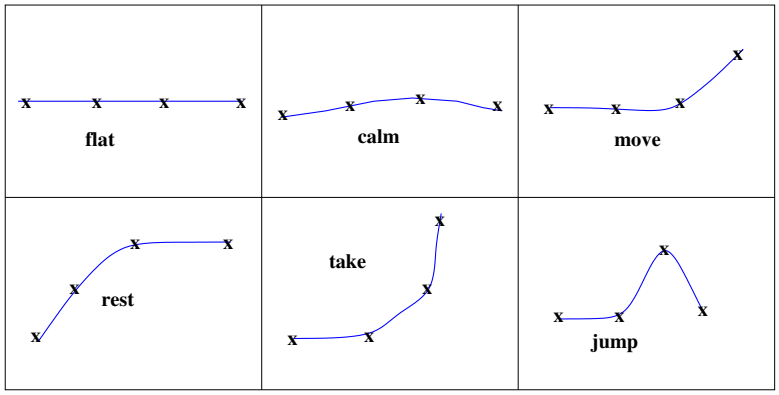

Figure A.32 – Illustration de la table des contextes A.6.

choix des deux seuils est relativement simple. Par conséquent, Le choix des seuils γ_0 et γ_{-1} est potentiellement crucial. Par exemple, choisir des valeurs fixes peut se révéler inefficace lorsque le signal présente une importante variation d'amplitude de l'échelle. Par conséquent, nous avons choisi de les calculer en ligne, sur la trame antérieure $\{u_{1-\omega},\ldots,u_{-3}\}$. Avec de trop faibles seuils, les extrapolations d'ordre élevé seraient rarement choisis, ce qui cause une perte des avantages de prédictions. En revanche, avec des seuils trop élevés, les sauts et les bruits ne seront pas en mesure d'être détectés. Comme les contextes sont basés sur des dérivées passées, nous avons choisi d'utiliser le point-milieu des valeurs absolues de l'estimateur statistique. Cela revient à définir : $\gamma_0 = \gamma_{-1} = \frac{1}{2}\max_{i\in[1-\omega,\ldots,-3]}(|u_i - u_{i+1}|)$.

Validation expérimentale

La simulation du modèle du moteur CFM partitionné est réalisée avec xMOD sur la même plate-forme que pour “RCosim”. De même, la simulation de référence est établie en respectant l'erreur relative Er, définie dans (A.6).

Pour explorer l'effet de l'extrapolation sur la précision, le pas de communication a été fixé à 250 µs dans une première série d'expériences. Cette valeur a été choisie pour fournir des résultats acceptables vis à vis de la précision ($Er \approx 1\,\%$), tout en étant assez large pour procéder à une extrapolation utile.

Les tests montrent qu'en utilisant seulement une prédiction polynomiale (sans contextes) sur le modèle moteur, la simulation échoue avec des erreurs d'intégration supérieures à celle de

la simulation de référence. Cela est dû à la nature hybride du modèle, par lequel les défaillances d'extrapolation sont causées par des discontinuités ainsi que des variations brusques de certaines variables à des instants spécifiques. Ces cas particuliers perdent tout le gain en précision, obtenu par des extrapolations réussites dans d'autres parties de la trajectoire des signaux. En revanche, en utilisant le prédicteur polynomial basé sur le contexte, les résultats de simulation sont presque toujours plus près de la trajectoire de référence que ceux calculés en considérant les entrées constantes pendant un pas de communication (voir figure A.33).

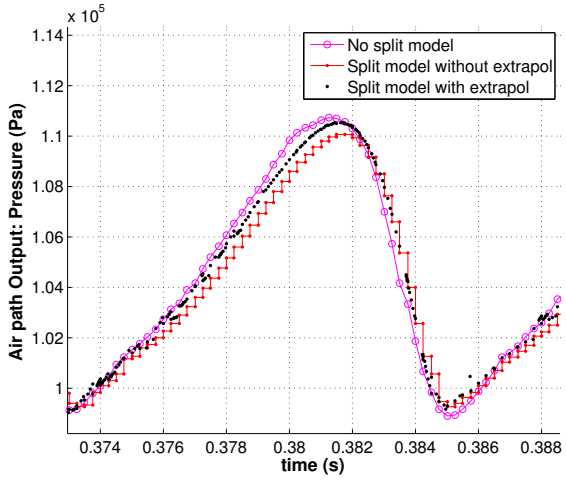

Figure A.33 – Sortie de la boucle d'air : pression.

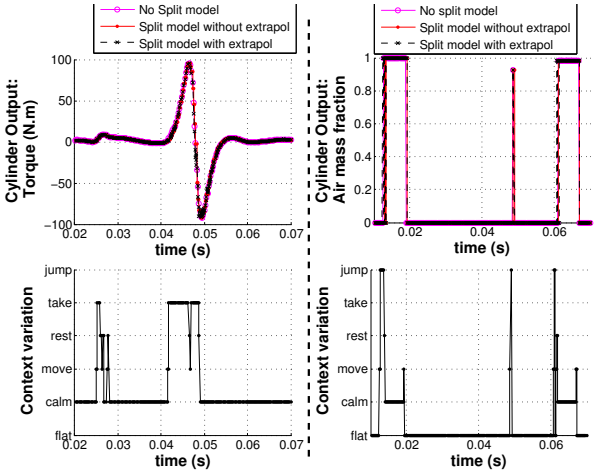

Figure A.34 – Evolution du contexte le long de la simulation.

La figure A.34 montre qu'en utilisant l'extrapolation basée sur le contexte, l'étape de prédiction est rejetée lorsque le signal présente un comportement discontinu, et le degré de prédiction est adapté en fonction de la pente du signal.

L'erreur d'intégration relative, cumulée le long d'une simulation assez longue, est représentée dans le tableau A.7. Le résultat montre que l'extrapolation basée sur le contexte diminue efficacement cette erreur pour les variables choisies, par exemple, de 63 % pour la température et de 72.5 % pour la densité du carburant. L'objectif ultime de l'extrapolation est de diminuer

Table A.7 – Erreur d'intégration relative.

Sorties	Er(%) sans extrapolation	Er(%) avec extrapolation
Pression	0.499	0.304
Température	0.511	0.19
Densité de l'air	0.784	0.31
Densité du carburant	3.55	0.978
Densité des gaz brulés	4.99	3.47

le temps de simulation en élargissant l'intervalle de synchronisation, tout en gardant l'erreur d'intégration relative Er inférieure aux limites prédéfinies. En effet, l'élargissement du pas de communication de 100 µs à 250 µs sans extrapolation (voir la figure A.35) permet de réduire le temps de calcul mais augmente en parallèle l'erreur (par exemple 6.97 % pour la densité des gaz brûlés et 340.5 % pour la densité du carburant). L'utilisation de l'extrapolation, pour le pas de communication 250 µs, permet heureusement de diminuer l'erreur à des valeurs proches de, ou en dessous de, celles mesurées pour le pas de communication 100 µs avec des entrées constantes. Le tableau A.8 montre une accélération de la simulation comparée à la référence à thread

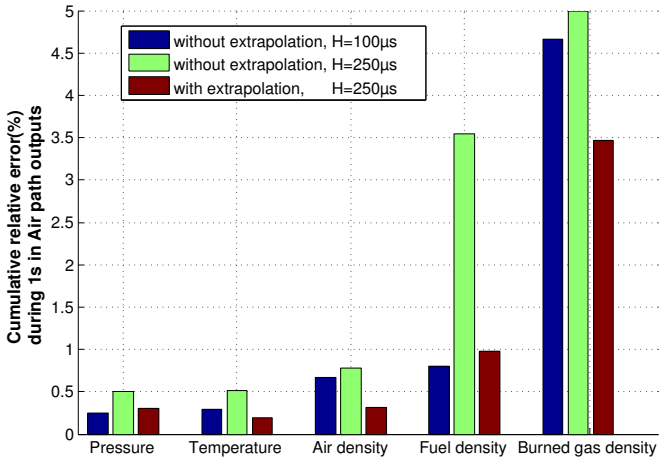

Figure A.35 – Erreur relative cumulée avec des pas de communication différents.

unique. Tout d'abord, il faut noter que lors de la décomposition du modèle en 5 "threads" qui sont intégrés en parallèle sur 5 coeurs, l'accélération est supra-linéaire par rapport au nombre de coeurs. En effet, le confinement de la détection et la localisation d'événements à l'intérieur de petits sous-systèmes permet une accélération des solveurs, assez importante pour sur-compenser les coûts du multi-threading. D'autre part, il semble que la combinaison de l'élargissement du pas de communication avec l'extrapolation basée sur le contexte, permet d'atteindre 10 % d'accélération supplémentaire sans perdre en précision. Encore plus surprenant, l'extrapolation permet même d'accélérer légèrement la simulation, qui peut être expliqué par le fait que la forme des entrées prédites permet une convergence plus rapide du pas du solveur.

Table A.8 – Accélération de la simulation.

Pas de communication	100 µs	250 µs	
Extrapolation	Non	Non	Oui
Accélération	8.9	10.01	10.07

Conclusion

Cette méthode permet d'élargir les pas de communication tout en gardant une précision d'intégration prédéfinie. Plutôt que d'utiliser des pas d'intégration et de communication très petits et très coûteux, nous proposons d'extrapoler le comportement des sous-modèles au cours des intervalles de synchronisation. Les résultats des tests sur le modèle moteur, qui est un système dynamique hybride, montre que l'extrapolation basée sur le contexte permet une accélération effective de la simulation avec des frais de calcul négligeables.

A.3 Conclusion générale

A.3.1 Conclusion

La complexité des systèmes mécatroniques est due à différents facteurs. Le développement d'un véhicule, par exemple, dépend d'un grand nombre de sous-systèmes tels que le groupe

motopropulseur, les carburants alternatifs, les systèmes de contrôle, les systèmes d'assistance au conducteur, la réglementation, la sécurité des véhicules, etc. La variété et la complexité des interactions entre ces composants nécessitent des échanges constants entre les différentes équipes d'experts.

L'approche de simulation système permet de projeter la complexité de l'ensemble du système dynamique avec son environnement au travers d'outils de développement virtuel, où les composants qui sont disponibles réellement en tant que matériels interagissent directement avec des modèles de systèmes. De plus, la co-simulation permet, à une étape précoce, de faire des prédictions et de prendre des décisions pour la conception de systèmes complexes ainsi que d'intégrer des systèmes temps-réel dans la simulation système.

Cependant, plusieurs questions relatives à la co-simulation doivent être résolues pour garantir la précision des résultats de simulation, tels que la précision des données échangées, le couplage de différents comportements des systèmes dynamiques, les contraintes temps-réel et la complexité de calcul des modèles.

De nos jours, les modèles phénoménologiques ne peuvent pas être simulés en HIL car ils sont très coûteux en temps de calcul. Pour répondre aux contraintes temps-réel, les ingénieurs passent beaucoup de temps à réduire la représentativité des modèles de la forme 0D phénoménologique à une forme simplifiée quasi-statique. L'objectif de cette thèse est d'améliorer cette phase de validation en gardant cette représentativité dans la simulation HIL. Pour cet objectif, nous proposons des méthodes qui permettent d'accélérer la simulation sans perdre en précision.

Les approches proposées et développées dans cette thèse sont structurées autour d'un cas d'étude d'un modèle 0D phénoménologique de moteur à combustion interne qui a été développé à IFP Energies nouvelles (chapitre 7). Tout d'abord une décomposition du modèle à partir d'un point de vue physique (chapitre 8) est présentée et appliquée dans le cadre de la co-simulation modulaire. L'approche du parallélisme au niveau du fil d'exécution "thread" montre l'intérêt de la décomposition des modèles quand il s'agit de systèmes hybrides complexes. En effet, les résultats des tests montrent que le coût majeur dans l'intégration numérique réside dans le calcul des dérivées et dans la détection et la localisation des événements. Par conséquent, ils peuvent être réduits grâce au découplage des sous-modèles.

Par la suite, dans le chapitre 9, une analyse de convergence des différents modèles de calcul utilisés pour la co-simulation modulaire dans le cadre de IFP Energies nouvelles (plus précisément dans le logiciel xMOD), est réalisée afin de déterminer les acteurs majeurs des erreurs de simulation. Nous montrons analytiquement que l'erreur est directement liée à la résolution numérique (pas d'intégration, ordre), le couplage des modèles et le pas de communication. L'effet des solveurs numériques est prouvé dans le chapitre 8.

Pour le couplage entre les modèles, nous proposons, dans le chapitre 10, une décomposition du modèle basée sur l'analyse structurelle du système, c'est-à-dire les différentes matrices d'incidence des états et des événements. Cette méthode est intéressante notamment pour les systèmes sans partition évidente ou intuitive. Ensuite, dans le chapitre 11, nous proposons une approche qui ordonnance d'une manière raffinée les différents fils d'exécution (modèles) sur les différents coeurs. Ce nouveau modèle de calcul ordonnance uniquement les opérations des variables d'entrée/sortie, puis exécute en parallèle le calcul des opérations des variables d'état (l'intégration numérique). Cette méthode montre, à travers le cas d'étude partitionné, une amélioration du temps de simulation par des accélérations supra-linéaires, déjà permises grâce à la co-simulation modulaire, et en même temps une amélioration de la précision de la simulation. De plus, cette approche permet de réduire le temps consacré par les ingénieurs à déterminer le bon partitionnement d'un système.

Enfin, pour permettre l'élargissement des pas de communication, nous proposons dans le chapitre 12, une prédiction polynomiale des entrées des modèles, basée sur des contextes prédéfinis. Les résultats montrent que l'utilisation de contextes est une importante valeur ajoutée pour l'extrapolation lorsqu'il s'agit de systèmes hybrides. En plus, grâce à cette nouvelle technique, les pas de communication entre les modèles faiblement couplés peuvent être relâchés et élargis pour accélérer la simulation et, en même temps, la précision des résultats peut être améliorée avec une extrapolation à coût négligeable.

A.3.2 Perspectives

Les axes de recherche suivants représentent des extensions possibles de ces travaux de thèse :

Pas de communication adaptatif

Dans les applications pratiques, les configurations actuelles de la co-simulation utilisent un pas de communication constant H, fixé par exemple par la spécification "FMI 1.0 for Model Exchange", étant donné qu'il n'y a pas de possibilité de reculs "rollbacks" (les variables d'états ne peuvent pas être sauvegardées). D'autres améliorations sont attendues avec des pas de communication adaptatifs, qui sont traités avec la version récente de la spécification "FMI 2.0 for Model Exchange". Il est prévu que l'adaptation du pas de communication permette de mieux gérer les différentes variations dynamiques des modèles [119].

En effet, la taille des pas de communication a un impact direct sur les erreurs de simulation (résumé dans la section 9.5), et pour avoir un contrôle efficace du pas de communication, il faudrait s'appuyer sur des estimations en ligne des erreurs induites par le relâchement du pas de communication (une première proposition a été détaillée dans la section 9.6). Effectivement, la stabilité des simulateurs multi-rythmes avec des pas adaptatifs doit être soigneusement évaluée, en se basant par exemple sur les récents travaux sur la propagation d'erreurs causée par la co-simulation modulaire [86].

Co-simulation multi-rythmes avec un ordonnancement à grains fins

La co-simulation avec un ordonnancement à grains fins " RCosim" (détaillée dans le chapitre 11), traite le cas où la co-simulation utilise un pas de communication de taille H qui est commun et partagé par tous les modèles. En effet, tous les modèles lisent leurs entrées et mettent à jour leurs sorties aux mêmes points de communication, qui sont multiples de H. Les futures améliorations visent à généraliser cette technique proposée pour le cas de la co-simulation multi-rythmes. En effet, cela permettra de profiter de l'avantage de deux approches 'multi-rythmes et " RCosim".

Extrapolation basée sur le contexte

Les travaux futurs visent à améliorer l'algorithme actuel d'extrapolation basée sur le contexte (détaillé dans chapitre 12), pour le rendre plus réactif au rafraîchissement des données et pour diminuer encore les erreurs de prédiction. Une autre possibilité serait d'ajouter un traitement aux signaux d'entrée, qui permet de les séparer en éléments plus simples, afin de prédire plus facilement avec les différents prédicteurs et de faire face au bruit. Quand il s'agit de polynômes,

les pré-processeurs d'ondelettes [120] pourrait être utile, comme ils jouent un rôle important dans l'ajustement des modèles polynomiaux.

Solveurs basés sur la quantification des variables d'état "QSS"

Cette thèse s'est intéressée aux solveurs numériques basés sur la discrétisation temporelle (voir la section 4.2). Par ailleurs, des investigations ont été également faites autour des solveurs basés sur la quantification des états (voir la section 4.3) afin de comparer leurs efficacités mutuelles, étant donné que les solveurs QSS sont connus par leur gestion efficace des systèmes discontinus [121]. Cependant, les résultats actuels intégrateurs utilisant une quantification des variables d'état (QSS Quantized State Solver) proviennent essentiellement d'exemples académiques, et nous avons rencontré des difficultés à les tester sur des exemples industriels tels que notre cas d'étude du modèle moteur. En effet, le principal obstacle était la traduction impérative du modèle du langage Modelica vers le langage μ-Modelica. Tout d'abord, cette traduction a été faite manuellement puisque le traducteur automatique est actuellement en cours de développement et donc pas encore disponible pour l'utilisateur final. Par exemple, pour la traduction en μ-Modelica du modèle moteur mono-cylindre, le nombre de lignes de codes est de l'ordre du millier. En second lieu, l'expressivité du langage μ-Modelica est limitée comparée à celle du langage Modelica, de sorte que le modèle moteur n'a pas pu être modélisé correctement avec μ-Modelica. Quoi qu'il en soit, les solveurs QSS semblent avoir un potentiel prometteur pour l'intégration des systèmes dynamiques avec de nombreuses discontinuités, et les progrès concernant la théorie des QSS et leurs outils associés méritent d'être suivis de près.

Épilogue

Comme dernier mot, ces travaux ont fourni des solutions efficaces et déjà utilisables pour relever les défis du sujet de la thèse. Ils contribuent aussi bien aux progrès scientifique que technologique. Les objectifs fixés au départ sont atteints, en fournissant un progrès méthodologique pour la co-simulation parallèle des systèmes complexes, ainsi que des solutions pratiques qui peuvent être utilisées directement par les ingénieurs.

Bien que notre cas d'étude présente un partitionnement naturellement évident et efficace, la méthodologie proposée peut être facilement appliquée à d'autres systèmes dynamiques hybrides. En effet, l'utilisation des solveurs à pas de temps variable, même dans un contexte temps-réel, représente la première étape clé au-delà de l'état de l'art sur le HIL. Comprendre quels sont les principaux goulots d'étranglement pour accélérer les solveurs, correspond à la deuxième étape clé, en fournissant les directions à suivre pour trouver des règles et des outils de partitionnement efficaces. Enfin, l'ordonnancement à grain fin et l'extrapolation permettent d'accélérer l'intégration numérique, de sorte que la simulation temps-réel de modèles à haut niveau de représentativité, dans le cadre de l'industrie automobile par exemple, devient possible.

Les méthodologies et les outils logiciels développés dans la thèse vont être rapidement exploités dans les développements industriels, notamment pour la conception de nouveaux moteurs à essence avec des niveaux d'émissions extrêmement faibles.

References

[1] C. Faure. *Real-time simulation of physical models toward hardware-in-the-loop validation.* PhD thesis, Univ. Paris-Est, France, 2011. (cited on pages 2, 59, 111, 125, and 149)

[2] 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, and J.-V. Peetz. The Functional Mockup Interface for tool independent exchange of simulation models. In *8th Int. Modelica Conf.*, Dresden, Germany, Mar 2011. Linköping Univ. Electronic Press. (cited on pages 3, 16, and 23)

[3] M. Valasek. *Simulation Techniques for Applied Dynamics*, chapter Modeling, simulation and control of mechatronical systems. Volume 507 of Arnold and Schiehlen [42], 2008. Courses and Lectures. (cited on pages 3 and 127)

[4] C. Faure, M. Ben Gaid, N. Pernet, M. Fremovici, G. Font, and G. Corde. Methods for real-time simulation of cyber-physical systems: Application to automotive domain. In *1st IEEE Workshop on CyPhy*, pages 1105–1110, Istanbul, Turkey, 2011. (cited on pages 4, 51, and 127)

[5] J. de Rosnay. Analytic vs. systemic approaches. Principia Cybernetica Web, 1997. (cited on page 9)

[6] F. Heylighen. Basic concepts of the systems approach. Principia Cybernetica Web, 1998. (cited on page 9)

[7] L. Gheorghe. *Continuous/Discrete co-simulation interfaces from formalization to implementation.* PhD thesis, École Polytechnique de Montréal, Canada, 2009. (cited on page 10)

[8] W. H. Kwon and S.-G. Choi. Real-time distributed software-in-the-loop simulation for distributed control systems. In *IEEE Int. Symp. on Computer Aided Control System Design CACSD*, pages 115–119, 1999. (cited on page 11)

[9] H. K. Fathy, Z. S. Filipi, J. Hagena, and J. L. Stein. Review of hardware-in-the-loop simulation and its prospects in the automotive area. In *Society of Photo-Optical Instrumentation Engineers (SPIE) Conf. Series*, volume 6228, 2006. (cited on page 11)

[10] S. Raman, N. Sivashankar, W. Milam, W. Stuart, and S. Nabi. Design and implementation of HIL simulators for powertrain control system software development. In *American Control Conf.*, volume 1, pages 709–713, 1999. (cited on page 11)

[11] F. E. Cellier and E. Kofman. *Continuous System Simulation.* Springer-Verlag New York, Inc., New York, 1st edition, 2006. (cited on pages 12, 29, 32, and 41)

[12] M. Falla. Advances in safety critical systems. Results and achievements from the DTI/EPSRC R&D programme in safety critical systems. Technical report, School of Computing and Communication - Lancaster Univ., UK, Jun 1997. (cited on page 13)

[13] G. Bernat, A. Burns, and A. Llamosi. Weakly hard real-time systems. *IEEE Trans. Comput.*, 50:308–321, 2001. (cited on page 13)

[14] P. J. Andrianiaina. *Robust control under slackened real-time constraints.* PhD thesis, Univ. de Grenoble, France, 2012. (cited on page 13)

[15] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, Jan 1973. (cited on pages 14 and 15)

[16] J. P. Lehoczky, L. Sha, and J. K. Strosnider. Enhanced aperiodic responsiveness in hard real-time environments. In *RTSS*, pages 261–270. IEEE Computer Society, 1987. (cited on page 15)

[17] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Trans. Comput.*, 39(9):1175–1185, Sep 1990. (cited on page 15)

[18] J. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. Anderson, and S. Baruah. A categorization of real-time multiprocessor scheduling problems and algorithms. In *Handbook on scheduling algorithms, methods, and models.* Chapman Hall/CRC, Boca, 2004. (cited on page 15)

[19] R. I. Davis and A. Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM Comput. Surv.*, 43(4):1–44, Oct 2011. (cited on page 15)

[20] J. A. Santos, G. Lima, K. Bletsas, and S. Kato. Multiprocessor real-time scheduling with a few migrating tasks. In *Real-Time Systems Symposium (RTSS), 2013 IEEE 34th*, pages 170–181, Dec 2013. (cited on page 15)

[21] A. Block, H. Leontyev, B. B. Brandenburg, and J. H. Anderson. A flexible real-time locking protocol for multiprocessors. In *RTCSA*, pages 47–56. IEEE Computer Society, 2007. (cited on page 15)

[22] T. Grandpierre and 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 System Design MEMOCODE*, pages 123–132, Mont Saint-Michel, France, 2003. (cited on pages 16, 101, and 144)

[23] T. Blochwitz, M. Otter, J. Åkesson, M. Arnold, C. Clauß, H. Elmqvist, M. Friedrich, A. Junghanns, J. Mauss, D. Neumerkel, H. Olsson, and A. Viel. Functional Mockup Interface 2.0: The standard for tool independent exchange of simulation models. In *9th Int. Modelica Conf.*, Munich, Germany, 2012. (cited on page 16)

[24] IEEE standard for Modeling and Simulation (M&S) High Level Architecture (HLA) – framework and rules. *IEEE Std 1516-2010 (Revision of IEEE Std 1516-2000)*, pages 1–38, Aug 2010. (cited on page 18)

[25] E. Noulard, J. Y. Rousselot, and P. Siron. CERTI, an open source RTI, why and how. In *Proceedings of the Spring Simulation Interoperability Workshop*, 2009. (cited on page 18)

[26] C. Gervais, J. Chaudron, P. Siron, R. Leconte, and D. Saussie. Real-time distributed aircraft simulation through HLA. In *Proceedings of 2012 IEEE/ACM 16th Int. Symp. on*

Distributed Simulation and Real Time Applications (DS-RT), pages 251–254, Oct 2012. (cited on page 18)

[27] G. Lasnier, J. Cardoso, P. Siron, C. Pagetti, and P. Derler. Distributed simulation of heterogeneous and real-time systems. In *Proceedings of the 2013 IEEE/ACM 17th Int. Symp. on Distributed Simulation and Real Time Applications (DS-RT)*, DS-RT '13, pages 55–62, Washington, DC, USA, 2013. IEEE Computer Society. (cited on page 18)

[28] J. B. Chaudron. *Architecture de simulation distribuée temps réel*. PhD thesis, Univ. de Toulouse, France, 2012. (cited on page 18)

[29] H. J. Hadj-Amor. *Contribution au prototypage virtuel de systèmes mécatroniques basé sur une architecture distribuée HLA. Expérimentation sous les environnements OpenModelica-OpenMASK*. PhD thesis, Univ. du Sud Toulon Var, France, 2008. (cited on page 18)

[30] M. U. Awais, P. Palensky, A. Elsheikh, E. Widl, and S. Matthias. The high level architecture RTI as a master to the Functional Mock-up Interface components. In *Int. Conf. on Computing, Networking and Communications (ICNC)*, pages 315–320, Jan 2013. (cited on page 19)

[31] C. Pantelides. The consistent initialization of Differential-Algebraic systems. *SIAM J. Sci. Stat. Comput.*, 9(2):213–231, 1988. (cited on page 22)

[32] J. Unger, A. Kroner, and W. Marquardt. Structural analysis of Differential-Algebraic Equation systems - theory and applications. *Comput. & Chem. Eng.*, 19(8):867–882, 1995. (cited on page 22)

[33] S. Chowdhry, H. Krendl, and A. Linninger. Symbolic numeric index analysis algorithm for Differential Algebraic Equations. *J. Am. Chem. Soc.*, 43(14):3886–3894, 2004. (cited on page 22)

[34] S. E. Mattsson and G. Söderlind. Index reduction in Differential-Algebraic Equations using dummy derivatives. *SIAM J. Sci. Comput.*, 14(3):677–692, May 1993. (cited on page 22)

[35] F. Zhang, M. Yeddanapudi, and P. Mosterman. Zero-crossing location and detection algorithms for hybrid system simulation. In *17th IFAC World Congress*, pages 7967–7972, Seoul, South Korea, 2008. (cited on pages 23 and 73)

[36] T. Bourke and M. Pouzet. Zélus: A synchronous language with ODEs. In *HSCC - 16th International Conference on Hybrid systems: Computation and control*, pages 113–118, Philadelphia, PA, USA, Apr 2013. ACM. (cited on page 24)

[37] Ch. Hylands, E. Lee, J. Liu, X. Liu, S. Neuendorffer, Y. Xiong, Y. Zhao, and H. Zheng. Overview of the Ptolemy project. Technical Report UCB/ERL M03/25, EECS Dept., Univ. of California, Berkeley, USA, 2003. (cited on page 25)

[38] S. L. Campbell, J.-P. Chancelier, and R. Nikoukhah. *Modeling and simulation in Scilab/Scicos.* Springer, New York, NY, USA, 2006. (cited on page 25)

[39] M. Najafi and R. Nikoukhah. Modeling and simulation of differential equations in Scicos. In *5th Int. Modelica Conf.*, Vienna, Austria, Sep 2006. (cited on page 25)

[40] E. Hairer and G. Wanner. *Solving Ordinary Differential Equations. II. Stiff and Differential-Algebraic problems*. Springer series in computational mathematics. Springer, Heidelberg, New York, 2010. (cited on page 28)

[41] C. F. Curtiss and J. O. Hirschfelder. Integration of stiff equations. *J. Proc. Nat. Acad. Sci.*, 38:235–243, 1952. (cited on page 28)

[42] M. Arnold and W. Schiehlen, editors. *Simulation Techniques for Applied Dynamics*, volume 507 of *CISM International Centre for Mechanical Sciences*. SpringerWienNewYork, 2008. Courses and Lectures. (cited on pages 29, 110, 157, and 164)

[43] U. M. Ascher and L. R. Petzold. *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. SIAM, Philadelphia, PA, USA, 1st edition, 1998. (cited on pages 29, 31, 32, 34, 35, and 37)

[44] R. LeVeque. *Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems (Classics in Applied Mathematics Classics in Applied Mathemat)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2007. (cited on page 30)

[45] A. C. Hindmarsh and L. R. Petzold. Algorithms and software for Ordinary Differential Equations and Differential-Algebraic Equations, part II: Higher-order methods and software packages. *Comput. Phys.*, 9:148–155, 1995. (cited on pages 30 and 38)

[46] P. D. Lax and R. D. Richtmyer. Survey of the stability of linear finite difference equations. *Commun. Pure Appl. Math.*, 9(2):267–293, 1956. (cited on page 31)

[47] K. K. Autar and E. K. Egwu. *Numerical Methods with Applications*. Lulu.com, 2008. (cited on page 34)

[48] R. P. Brent. *Algorithms for Minimization Without Derivatives*. Prentice-Hall, Inc., Englewood Cliffs, New Jersery, 1973. (cited on page 34)

[49] P. McEvoy. *Classical Theory*. Theory of interacting systems. MicroAnalytix, 2002. (cited on page 35)

[50] N. Ozisik. *Finite Difference Methods in Heat Transfer, Second Edition*. Heat Transfer. Taylor & Francis, 1994. (cited on page 35)

[51] K. Radhakrishnan and A. C. Hindmarsh. Description and use of LSODE, the Livermore Solver for Ordinary Differential Equations. Technical Report UCRL-ID-113855, Lawrence Livermore National Laboratory LLNL, CA, USA, 1994. (cited on page 38)

[52] L. Petzold. Automatic selection of methods for solving stiff and nonstiff systems of Ordinary Differential Equations. *SIAM J. on Sci. Stat. Comput.*, 4(1):136–148, 1983. (cited on page 38)

[53] L. R. Petzold. A description of DASSL: A differential/algebraic system solver. In *International Mathematics and Computers Simulation congress on Systems Simulation and Scientific computing*, Montreal, Canada, Aug 1982. (cited on page 38)

[54] K. E. Brenan, S. L. Campbell, and L. R. Petzold. *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*. Classics in Applied Mathematics. Society for Industrial and Applied Mathematics, 1996. (cited on page 38)

[55] B. P. Zeigler, T. G. Kim, and H. Praehofer. *Theory of modeling and simulation : Integrating discrete event and continuous complex dynamic systems*. Academic Press, Amsterdam, San Diego (Calif.), London, 2000. (cited on pages 39 and 40)

[56] E. Kofman and S. Junco. Quantized-state systems: A DEVS approach for continuous system simulation. *Trans. Soc. Comput. Simul. Int.*, 18(3):123–132, Sep 2001. (cited on pages 39, 43, 92, and 138)

[57] E. Kofman. Discrete event simulation of hybrid systems. *SIAM J. Sci. Comput.*, 25:1771–1797, 2004. (cited on page 41)

[58] E. Kofman. Relative error control in quantization based integration. *Lat. Am. Appl. Res.*, 39:231–237, Jul 2009. (cited on page 43)

[59] E. Kofman. A second-order approximation for DEVS simulation of continuous systems. *Simulation: Trans. Soc. Model. Simul. Int.*, 78(2):76–89, 2002. (cited on page 43)

[60] E. Kofman. A third order discrete event simulation method for continuous system simulation. *Lat. Am. Appl. Res.*, 36(2):101–108, 2006. (cited on page 43)

[61] G. Migoni, E. Kofman, and F. E. Cellier. Integración por cuantificación de sistemas stiff. *Revista Iberoamericana de Automàtica e Informática Industrial*, 4(3):97–106, 2007. (cited on page 44)

[62] G. Migoni and E. Kofman. Linearly implicit discrete event methods for stiff ODEs. *Lat. Am. Appl. Res.*, 39:245–254, Jul 2009. (cited on page 44)

[63] F. E. Cellier, E. Kofman, G. Migoni, and M. Bortolotto. Quantized state system simulation. In *Summer Simulation Multiconference*, Edinburgh, Scotland, 2008. (cited on page 44)

[64] E. Kofman and B. Zeigler. DEVS simulation of marginally stable systems. In *17th IMACS World Congress*, Paris, France, 2005. (cited on page 44)

[65] G. Migoni, E. Kofman, and F. E. Cellier. Quantization-based new integration methods for stiff ODEs. *Simulation: Trans. Soc. Model. Simul. Int.*, 88(4):387–407, 2012. (cited on page 44)

[66] K. Burrage. *Parallel and Sequential Methods for Ordinary Differential Equations*. Oxford Science Publications, 1995. (cited on page 48)

[67] K. Burrage. Parallel methods for initial value problems. *J. Appl. Numer. Math.*, 11(1-3):5–25, 1993. (cited on page 48)

[68] A. Iserles and S. P. Nørsett. On the theory of parallel Runge-Kutta methods. *IMA J. Numer. Anal.*, 10(4):463–488, 1990. (cited on page 48)

[69] W. L. Miranker and W. Liniger. Parallel methods for the numerical integration of Ordinary Differential Equations. *J. Math. Comput.*, 21(99):303–320, 1967. (cited on page 48)

[70] K. R. Jackson. A survey of parallel numerical methods for initial value problems for Ordinary Differential Equations. *IEEE Trans. Magn.*, 27(5):3792–3797, 1991. (cited on page 48)

[71] P. J. van der Houwen and B. P. Sommeijer. Parallel iteration of high-order Runge-Kutta methods with stepsize control. *J. Comput. Appl. Math.*, 29:111–127, 1990. (cited on page 48)

[72] G. D. Byrne and A. C. Hindmarsh. PVODE, an ODE solver for parallel computers. *Int. J. High Perform. Comput. Appl.*, 13(4):254–365, 1999. (cited on page 48)

[73] J. Clauberg and H. Ulbrich. An adaptive internal parallelization method for multibody simulations. In *12th Pan-American Congress of Applied Mechanics*, Port of Spain, Trinidad, 2012. (cited on page 48)

[74] G. Horton and S. Vandewalle. A space-time multigrid method for parabolic PDEs. Technical report, Univ. Erlangen, Germany, 1993. (cited on page 49)

[75] J. Lions, Y. Maday, and G. Turinici. Résolution d'EDP par un schéma en temps "pararéel". *Comptes Rendus de l'Académie des Sciences - Series I - Mathematics*, 332(7):661–668, 2001. (cited on page 49)

[76] C. Farhat and M. Chandesris. Time-decomposed parallel time-integrators: Theory and feasibility studies for fluid, structure, and fluid-structure applications. *Int. J. Numer. Meth. Eng.*, 58(9):1397–1434, 2003. (cited on page 49)

[77] P. Deuflhard. *Newton Methods for Nonlinear Problems*, volume 35 of *Springer Series in Computational Mathematics*. Springer, 2004. (cited on page 49)

[78] D. Guibert. *Analyse de méthodes de résolution parallèles d'EDO/EDA raides*. PhD thesis, Univ. Claude Bernard - Lyon I, France, 2009. (cited on page 49)

[79] E. Lelarasmee, A. E. Ruehli, and 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(3):131–145, Jul 1982. (cited on page 49)

[80] K. Burrage, C. Dyke, and B. Pohl. On the performance of parallel waveform relaxations for differential systems. *J. Appl. Numer. Math.*, 20:39–55, 1996. (cited on pages 49 and 50)

[81] S. Y. R. Hui and 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(6):379–384, 1990. (cited on page 50)

[82] B. Eriksson, P. Nordin, and P. Krus. Hopsan NG, a C++ implementation using the TLM simulation technique. In *51th Conf. on Simulation and Modelling*, Oulu, Finland, 2010. (cited on page 51)

[83] M. Sjölund, R. Braun, P. Fritzson, and P. Krus. Towards efficient distributed simulation in Modelica using Transmission Line Modeling. In *3rd Int. Workshop on Equation-Based Object-Oriented Languages and Tools EOOLT*, pages 71–80, Oslo, Norway, 2010. Linköping Univ. Electronic Press. (cited on page 51)

[84] R. Braun. *Multi-threaded distributed system simulations: Using bi-lateral delay lines*. Licentiate thesis, Linköping Univ., Sweden, 2013. (cited on page 51)

[85] R. Braun and P. Krus. Multi-threaded real-time simulations of fluid power systems using transmission line elements. In *proceeding of 8th Int. Fluid Power Conf.*, Dresden, Germany, 2012. (cited on page 51)

[86] M. Arnold. Stability of sequential modular time integration methods for coupled multibody system models. *J. Comput. Nonlin. Dyn.*, 5(3), 2010. (cited on pages 51, 122, and 155)

[87] M. Ben Gaïd, G. Corde, A. Chasse, B. Léty, R. De La Rubia, and 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. (cited on page 51)

[88] A. Ben Khaled, M. Ben Gaïd, D. Simon, and G. Font. Multicore simulation of powertrains using weakly synchronized model partitioning. In *IFAC Workshop on Engine and Powertrain Control Simulation and Modeling ECOSM*, pages 448–455, Rueil-Malmaison, France, 2012. (cited on page 51)

[89] A. Ben Khaled, M. Ben Gaïd, and D. Simon. Parallelization approaches for the time-efficient simulation of hybrid dynamical systems: Application to combustion modeling. In *5th Int. Workshop on Equation-Based Object-Oriented Languages and Tools EOOLT*, pages 27–36, Nottingham, UK, 2013. Linköping Univ. Electronic Press. (cited on page 51)

[90] F. Casella. A strategy for parallel simulation of declarative object-oriented models of generalized physical networks. In *5th Int. Workshop on Equation-Based Object-Oriented Languages and Tools EOOLT*, pages 45–51, Nottingham, UK, 2013. Linköping Univ. Electronic Press. (cited on page 51)

[91] A. V. Papadopoulos and A. Leva. Automating dynamic decoupling in object-oriented modelling and simulation tools. In *5th Int. Workshop on Equation-Based Object-Oriented Languages and Tools EOOLT*, Nottingham, UK, 2013. Linköping Univ. Electronic Press. (cited on page 51)

[92] R. E. Bryant. Simulation of packet communication architecture computer systems. Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 1977. (cited on page 52)

[93] D. R. Jefferson. Virtual time. *ACM Trans. Program. Lang. Syst.*, 7(3):404–425, Jul 1985. (cited on page 52)

[94] D. M. Rao, N. V. Thondugulam, R. Radhakrishnan, and P. A. Wilsey. Unsynchronized parallel discrete event simulation. In *Proceedings of the 30th Conf. on Winter simulation*, WSC '98, pages 1563–1570, Los Alamitos, CA, USA, 1998. IEEE Computer Society Press. (cited on page 52)

[95] F. Bergero, E. Kofman, and F. E. Cellier. A novel parallelization technique for DEVS simulation of continuous and hybrid systems. *Simulation: Trans. Soc. Model. Simul. Int.*, 2012. (cited on pages 52 and 53)

[96] J. B. Heywood. *Internal combustion engine fundamentals*. McGraw-Hill series in mechanical engineering. McGraw-Hill, 1988. (cited on page 60)

[97] I. I. Wiebe. *Brennverlauf und Kreisprozess von Verbrennungsmotoren*. VEB Verlag Technik, Berlin, 1970. (cited on pages 62 and 128)

[98] S. Richard, S. Bougrine, G. Font, F.-A. Lafossas, and F. Le Berr. On the reduction of a 3D CFD combustion model to build a physical 0D model for simulating heat release, knock and pollutants in SI engines. *Oil & Gas Science and Technology*, 64:223–242, 2009. (cited on pages 62 and 128)

[99] O. Colin, A. Benkenida, and C. Angelberger. A 3D modeling of mixing, ignition and combustion phenomena in highly stratified gasoline engines. *Oil & Gas Science and Technology*, 58:47–62, 2003. (cited on pages 62 and 128)

[100] S. Richard, G. Font, F. Le Berr, O. Grasset, and M. Fremovici. On the use of system simulation to explore the potential of innovative combustion systems: Methodology and application to highly downsized SI engines running with ethanol-gasoline blends. *JSAE Paper*, 2011. (cited on pages 63 and 128)

[101] Z. Benjelloun-Touimi, M. Ben Gaïd, J. Bohbot, A. Dutoya, H. Hadj-Amor, P. Moulin, H. Saafi, and 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, Mar 2011. Linköping Univ. Electronic Press. (cited on pages 63 and 128)

[102] P. Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica*. Wiley-IEEE Computer Society Pr, 2003. (cited on pages 63 and 128)

[103] T. Blochwitz, M. Otter, M. Arnold, C. Bausch, C. Clauß, H. Elmqvist, A. Junghanns, J. Mauss, M. Monteiro, T. Neidhold, D. Neumerkel, H. Olsson, J.-V. Peetz, and S. Wolf. The Functional Mockup Interface for tool independent exchange of simulation models. In *8th Int. Modelica Conf.*, Dresden, Germany, Mar 2011. Linköping Univ. Electronic Press. (cited on pages 64 and 128)

[104] A. Viel. Implementing stabilized co-simulation of strongly coupled systems using the Functional Mock-up Interface 2.0. In *10th Int. Modelica Conf.*, Lund, Sweden, 2014. (cited on pages 74 and 135)

[105] M. Arnold. *Simulation Techniques for Applied Dynamics*, chapter Numerical methods for simulation in applied dynamics. Volume 507 of Arnold and Schiehlen [42], 2008. Courses and Lectures. (cited on pages 76, 115, and 150)

[106] G. Karypis and V. Kumar. MeTiS: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. Technical report, Univ. of Minnesota, Dept. of Comput. Sci., USA, 1998. (cited on page 89)

[107] Ü. V. Çatalyürek. *Hypergraph models for sparse matrix partitioning and reordering*. PhD thesis, Bilkent Univ., Turkey, 1999. (cited on pages 91 and 137)

[108] M. C. Ferris and J. D. Horn. Partitioning mathematical programs for parallel solution. *J. Math. Programming*, 80:35–61, 1998. (cited on page 91)

[109] F. Bergero, X. Floros, J. Fernández, E. Kofman, and F. E. Cellier. Simulating Modelica models with a stand-alone quantized state system solver. In *9th Int. Modelica Conf.*, Munich, Germany, 2012. (cited on pages 92 and 138)

[110] M Sjölund. *Tools for Understanding, Debugging, and Simulation Performance Improvement of Equation-Based Models*. Licentiate thesis, Linköping Univ., 2013. (cited on page 99)

[111] M. J. Weinberger, G. Seroussi, and G. Sapiro. The LOCO-I lossless image compression algorithm: Principles and standardization into JPEG-LS. *IEEE Trans. Image Process.*, 9(8):1309–1324, 2000. (cited on pages 110 and 150)

[112] N. Wiener. *Extrapolation, interpolation, and smoothing of stationary time series*. Cambridge, Technology Press of Massachusetts Institute of Technology, and New York, Wiley, 1949. (cited on page 111)

[113] R. G. Brown. *Smoothing, Forecasting and Prediction of Discrete Time Series*. Prentice-Hall, 1962. (cited on page 111)

[114] G. E. P. Box, G. M. Jenkins, and G. C. Reinsel. *Time Series Analysis: Forecasting and Control*. Probability and Statistics. Wiley, 2008. (cited on page 111)

[115] R. W. Hamming. *Numerical methods for scientists and engineers*. Dover publications, 1973. (cited on pages 111 and 149)

[116] S. M. Stigler. Gauss and the invention of least squares. *J. Ann. Stat.*, 9(3):465–474, 1981. (cited on page 112)

[117] G. F. C. de Bruyn and J. M. de Villiers. Formulas for $1+2^p+3^p+\ldots+n^p$. *Fibonacci Q.*, 32(3):271–276, 1994. (cited on page 112)

[118] M. Arnold, C. Clauß, and T. Schierz. Error analysis and error estimates for co-simulation in FMI for model exchange and co-simulation V2.0. *Arch. Mech. Eng.*, LX(1):6–156, 2013. (cited on page 119)

[119] T. Schierz, M. Arnold, and C. Clauß. Co-simulation with communication step size control in an FMI compatible master algorithm. In *9th Int. Modelica Conf.*, Munich, Germany, 2012. (cited on pages 122 and 155)

[120] C. Chaux, J.-C. Pesquet, and L. Duval. Noise covariance properties in dual-tree wavelet decompositions. *IEEE Trans. Inform. Theory*, 53(12):4680–4700, Dec 2007. (cited on pages 123 and 156)

[121] G. Migoni, M. Bortolotto, E. Kofman, and F. E. Cellier. Linearly implicit quantization-based integration methods for stiff Ordinary Differential Equations. *Simul. Model. Pract. Th.*, 35(0):118–136, 2013. (cited on pages 123 and 156)

List of publications

Articles in journals

- **A. Ben Khaled**, M. Ben Gaïd, N. Pernet, and D. Simon. Fast multi-core co-simulation of Cyber-Physical Systems: Application to internal combustion engines. *Simulation Modelling Practice and Theory*, Elsevier. Forthcoming 2014.

Proceedings of international conferences with reviewing committee

- **A. Ben Khaled**, L. Duval, M. Ben Gaïd, and D. Simon. Context-based polynomial extrapolation and slackened synchronization for fast multi-core simulation using FMI. In *10th International Modelica Conference*, Lund, Sweden, 2014. `http://www.ep.liu.se/ecp/096/023/ecp14096023.pdf`.

- **A. Ben Khaled**, M. Ben Gaïd, and 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)*, pages 27-36, Nottingham, UK, 2013. Linköping Univ. Electronic Press, `http://www.ep.liu.se/ecp/084/004/ecp13084004.pdf`.

- **A. Ben Khaled**, M. Ben Gaïd, D. Simon, and G. Font. Multicore simulation of powertrains using weakly synchronized model partitioning. In *IFAC Workshop on Engine and Powertrain Control Simulation and Modeling (E-COSM)*, pages 448-455, Rueil-Malmaison, France, 2012, `http://www.ifac-papersonline.net/Detailed/56925.html`.

- **A. Ben Khaled**, M. Ben Gaïd, and D. Simon. Towards a weakly-hard approach for real-time simulation. In *5th Junior Researcher Workshop on Real-Time Computing (JRWRTC) in conjunction with 19th International Conference Real-Time and Network Systems (RTNS)*, Nantes, France, 2011, `http://rtns2011.irccyn.ec-nantes.fr/files/jrwrtc2011.pdf`.

UNIVERSITÉ DE GRENOBLE

Résumé

De nos jours, la validation des unités de contrôle électronique ECU se fonde généralement sur la simulation Hardware-In-the-Loop où les systèmes physiques qui manquent sont modélisés à l'aide des équations différentielles hybrides. La complexité croissante de ce type de modèles rend le compromis entre le temps de calcul et la précision de la simulation difficile à satisfaire. Cette thèse étudie et propose des méthodes d'analyse et d'expérimentation destinées à la co-simulation temps-réel ferme de modèles dynamiques hybrides. Elle vise notamment à définir des solutions afin d'exploiter plus efficacement le parallélisme fourni par les architectures multi-cœurs en utilisant de nouvelles méthodes et paradigmes de l'allocation des ressources. La première phase de la thèse a étudié la possibilité d'utiliser des méthodes d'intégration numérique permettant d'adapter l'ordre comme la taille du pas de temps ainsi que de détecter les événements et ceci dans le contexte de la co-simulation modulaire avec des contraintes temps-réel faiblement dures. De plus, l'ordre d'exécution des différents modèles a été étudié afin de démontrer l'influence du respect des dépendances de données entre les modèles couplés sur les résultats de la simulation. Nous avons proposé pour cet objectif, une nouvelle méthode de co-simulation qui permet le parallélisme complet entre les modèles impliquant une accélération supra-linéaire sans pour autant ajouter des erreurs liées à l'ordre d'exécution. Enfin, les erreurs de retard causées par la taille de pas de communication entre les modèles ont été améliorées grâce à une nouvelle méthode d'extrapolation par contexte des signaux d'entrée. Toutes les approches proposées visent de manière constructive à améliorer la vitesse de simulation afin de respecter les contraintes temps-réel, tout en gardant la qualité et la précision des résultats de simulation sous contrôle. Ces méthodes ont été validées par plusieurs essais et expériences sur un modèle de moteur à combustion interne et intégrées à un prototype du logiciel xMOD.

Mots-clés : Simulation temps-réel, Functional Mockup Interface (FMI), simulation multi-cœurs, Systèmes Cyber-Physiques, Groupe motopropulseurs (GMP), ordonnancement.

Abstract

Nowadays the validation of Electronic Control Units ECUs generally relies on Hardware-in-The-Loop simulation where the lacking physical systems are modeled using hybrid differential equations. The increasing complexity of this kind of models makes the trade-off between time efficiency and the simulation accuracy hard to satisfy. This thesis investigates and proposes some analytical and experimental methods towards weakly-hard real-time co-simulation of hybrid dynamical models. It seeks in particular to define solutions in order to exploit more efficiently the parallelism provided by multi-core architectures using new methods and paradigms of resource allocation. The first phase of the thesis studied the possibility of using step-size and order control numerical integration methods with events detection in the context of real-time modular co-simulation when the time constraints are considered weakly-hard. Moreover, the execution order of the different models was studied to show the influence of keeping or not the data dependencies between coupled models on the simulation results. We proposed for this aim a new method of co-simulation that allows the full parallelism between models implying supra-linear speed-ups without adding errors related to their execution order. Finally, the delay errors due to the communication step-size between the models were improved thanks to a proposed context-based inputs extrapolation. All proposed approaches target constructively to enhance the simulation speed for the compliance to real-time constraints while keeping the quality and accuracy of simulation results under control and they are validated through several test and experiments on an internal combustion engine model and integrated to a prototype version of the xMOD software.

Keywords: Real-time simulation, Functional Mock-up Interface (FMI), multi-core simulation, Cyber Physical System, powertrain, scheduling.